Full Potential Local Orbital Minimum Basis Bandstructure Scheme

User's Manual

by

Klaus Koepernik

March 11, 2014

Contents

List of Tables VII

Nomenclature IX

1 Introduction 1
- 1.1 Installation procedures . . . 2
- 1.2 Program structure . . . 9
- 1.3 Input structure . . . 11

2 Version control 12
- 2.1 Upgrading . . . 12
- 2.2 Downgrading . . . 13

3 Files 15
- 3.1 General . . . 15
- 3.2 File class 1 . . . 16
- 3.3 File class 2 . . . 22
- 3.4 File class 3 . . . 27
- 3.5 Directories . . . 28

4 Programs 29
- 4.1 LSDA+U . . . 29
- 4.2 Grep data from output: GREPFPLO . . . 30
  - 4.2.1 Specific data . . . 30
  - 4.2.2 Categorized data . . . 32
- 4.3 Adding band weights: FADDWEI . . . 33
- 4.4 Graphical interface: XFPLO . . . 37
  - 4.4.1 Introduction . . . 37
  - 4.4.2 Trouble shooting . . . 38
  - 4.4.3 Hotkeys . . . 38
  - 4.4.4 The 3D View . . . 38
  - 4.4.5 Atom distances and angles . . . 39
  - 4.4.6 Symmetry Dialog . . . 39
    - 4.4.6.1 Groups . . . 39
    - 4.4.6.2 Wyckoff positions . . . 40

4.4.7 Boundary Dialog . . . 40
4.4.8 Bond Dialog . . . 41
4.4.9 Bond-Poly Dialog . . . 42
4.4.10 Find atoms Dialog . . . 42
4.4.11 Fermi energy and bands . . . 42
4.4.12 Brillouin zones: High symmetry points . . . 43
4.4.12.1 monoclinic lattices . . . 43
4.4.13 Mesh Dialog . . . 43
4.4.14 Fat band editor . . . 44
4.4.15 Band Weight Contrib Dialog . . . 45
4.4.16 Unfold Editor . . . 45
4.4.17 Annotations . . . 46
4.4.17.1 Axes . . . 46
4.4.17.2 Display cells . . . 46
4.4.18 Fog Dialog . . . 46
4.4.19 Atom properties . . . 47
4.4.20 Printing . . . 47
4.4.21 Preferences . . . 48
4.4.21.1 Graphics . . . 48
4.4.21.2 External . . . 48
4.4.21.3 OpenGL . . . 49
4.4.22 Manage executables . . . 49
4.4.23 Themes . . . 49
4.4.24 Lighting . . . 50
4.4.25 Molecular/individual band weights . . . 51
4.4.26 Structure viewer . . . 52
4.4.27 Fermi surface viewer . . . 52
4.4.28 Density mapper . . . 53
4.5 Plotting program: XFBP . . . 54
4.5.1 Scripting . . . 54
4.5.2 GUI . . . 82
4.5.2.1 Plotting window . . . 82
4.5.2.2 Scripting window . . . 83
4.5.3 Logarithmic plots . . . 83
4.5.4 Text formating . . . 84

4.5.5 Data . . . 86
4.5.6 Files . . . 86
4.5.7 Command line options . . . 86
4.5.7.1 File type flags . . . 86
4.5.8 Data file types . . . 87
4.6 Occupation matrix manipulator: DMATEDIT . . . 87
4.7 Optics data: FOPTICS . . . 89
4.7.1 Optical functions . . . 93
4.7.1.1 General . . . 93
4.7.1.2 Intra-band (Drude) . . . 94
4.7.1.3 Intra-band and constant interband ε_∞ . . . 95

5 Automation, scripting, pipe-mode 97
5.1 Rules . . . 97
5.2 Syntax . . . 99
5.3 How to set up automation . . . 101
5.4 Advanced features . . . 103
5.4.1 Initial polarization, inital spin split . . . 103
5.5 Example . . . 105

Appendix 115
A Summary of Changes . . . 115
B FAQ . . . 118

Index 121

List of Tables

3.1 Band weights files. 24

4.1 Module output prefixes 33
4.2 Mapping spherial harmonics to functions. 34
4.3 Axes hotkeys 38
4.4 Boundary/Display cell hotkeys 38
4.5 Optical functions 91

Nomenclature

`stdout` the unix standard output file/channel

`stderr` the unix standard error file/channel

`stdin` the unix standard input file/channel

BZ Brillouin zone

CPA coherent crystal approximation

DFT density functional theory

FSM fixed spin moment

GGA generalized gradient approximation

LO local orbital

PID the unix process ID

VCA virtual crystal approximation

xc exchange and correlation

(L)DOS (local/projected) density of states

L(S)DA+U local (spin) density approximation + U-functional

L(S)DA local (spin) density approximation

v^{cf} Confining potential

v^{cryst} Effective crystall potential

Chapter 1

Introduction

This document shall help to understand the FPLO package. We adopt the following notions. Text in typewriter style refers to unix commands, FEDIT options, file content and such things. Text in blue typewriter are names of files (e.g. `=.in`). The special symbols FPLO, FEDIT, ... are placeholders for the actual (version-related) full names of these programs (if applicable).

Example: You have installed the binary `fplo14.00-45` and related binaries. Then the unix command (example)

```
FEDIT  -pipe < ./=.pipefile 2>./+log 1>/dev/null
```

means

```
fedit14.00-45 -pipe < ./=.pipefile 2>./+log 1>/dev/null
```

for your installation.

In order to avoid confusion, the installation process **discontinued** to create links with the generic names `fplo`, `fedit` and so on after version 4 or so. The new policy is to call the programs with their full names. Beware, that the full name may include a platform specifier and even a user defined build branch name to use multiple architectures with a common file system[1].

Hint: Every important output produced by the FPLO package contains the version number of the program, which produced the output.

External materials, mostly additional information files and help screens from program parts are marked with a bar at the side, like for this paragraph. Help

[1] There is an exception, the program FPIOTEST, which is used to manipulate input files on a basic level, usually from inside maintenance scripts. This program is still installed with a generic name, since we need the generic name in order for that scripts to work. And furthermore the program usually does not change in between versions.

1.1 Installation procedures

Copy of your-installation-path/FPLO/FPLO14.../install/README:

```
Installation of FPLO-14.00-45
Ulrike Nitzsche (u.nitzsche@ifw-dresden.de)
Nov. 2013

I.Prerequisites
===============
What do you need to get FPLO-14.00-45 running?

At least a Unix system. FPLO-14.00-45 is tested under Linux on i386, x86_64
and ia64 architecture with the ifort compiler and it is planned to port
it to the gfortran compiler. Therefore the code should run (in near future)
on most Unix platforms with gfortran and gcc. (Test show that gfortran is not
as runtime efficient as ifort.) It is not planned to port
FPLO-14.00-45 to any Windows flavour.

The sources of FPLO-14.00-45 are a mix of C, C++ and F90 code.
Therefore you need an ANSI conform C/C++-compiler and an F90-compiler.
For commercial Unix systems (AIX, HPUX, Solaris) we suggest to use
the native compilers to get best performance. Please take care to use latest
versions and bug fixes.
Unfortunately we are not able to test all the platforms and compiler
versions. Therefore we can only support a gfortran/gcc or ifort/gcc
 installation there.

For all Linux (even for x86_64 Opteron) systems we recommend to
use gcc and ifort10 (ifort is the Intel Fortran compiler, after a registration
procedure for academic use you can get a unsupported non-commercial version
from http://www.intel.com/support/performancetools/fortran/linux/
(this link might have changed)
).

For the graphical software comming with FPLO you need to install the
qt libraries including development libraries. You need Qt version >=4.6
but < 5 in the moment. Ports to Qt5 will happen once qt5 is becomming
the standard.

There are some useful add-ons to FPLO-14.00-45. To use them you might need
 perl5-Interpreter (for developing the code and some auxillary tools)
```

```
To unpack the source tree you need tar and gzip. To use the installation
scripts 'make' is required. On HPUX, we recommend to use gmake (gnu-make)
instead of HP's own make (for a detailed explanation see
http://www.ifw-dresden.de/FPLO/faq.htm).

II. Get the source  (TODO)
==========================
To get the source follow the instructions at
http://www.ifw-dresden.de/FPLO/conditions.htm. After signing the license
conditions and transfering the license fee you will get the source
of FPLO-14.00-45 as MIME-attachment FPLO14.00-45.tar.gz, another file
ftreeinst.sh, and this README by email. Save these 3 files
to disk and move them to your install directory. Let's say, you want to
install FPLO in your home directory. Hence do:

  mv FPLO14.00-45.tar.gz ftreeinst.sh README $HOME

Change into your install directory (in our example the home directory):

  cd $HOME

Now we can start the installation procedure.

Starting with FPLO-3 we prepared our installation scripts for parallel
installation of several FPLO-versions (including subversions) side by side.
Therefor all FPLO-versions are located in a directory FPLO in subdirectories
FPLO-<Version>. In the directories FPLO/FPLO-<Version> you can
view the source code, change the code and compile it. The binaries
are installed into FPLO/bin as fplo-<version>, fedit-<version> and so on.
To create the necessary directory tree, you have to start the installation
with

  sh ./ftreeinst.sh

If you don't have a directory called FPLO no problems are expected, if there
is a directory with this name you are asked to rename it.

Congratulations! The first step of the installation procedure is finished.

Now you will find a directory FPLO14.00-45 in your FPLO directory.
Change to this directory:
```

```
  cd FPLO/FPLO14.00-45

Here, you find a directory "install". Change to it:

  cd install

Now, you have to choose the type of installation:

  If you use Linux with gcc and ifort or gfortran with default options
go to III. Simple Installation on Linux with ifort as F90-Compilers .


To install xfplo and xfbp please go to

   cd XFPLO_rel

and follow the instructions in the README file.
And

   cd XFBP_rel

and follow the instructions in the README file.



  For installation of multiple builds of the same FPLO version go to IIIa.

  In all other cases go to IV. Advanced Installation .

III. Simple Installation on Linux with ifort or gfortran as F90-Compilers
=========================================================================

You are in the directory FPLO14.00-45/install .

To create the appropriate Makefile for your architecture, type:

  sh ./MMakefile

You will be asked if f90 and gcc/g++ are your favorite compilers.

Answer the first question with either ifort or gfortran and the second/third questi
with [enter].
```

```
If ifort is used it comes with the MKL library. The next question is if you
want to use the MKL-librarie's eigenvalue solver. Try answering y[es].
The installation tries to find the mkl path. If it does go on.
If it does not find it you will have to setup the linkage yourself or
restart MMakefile and answer n[o], which results in using FPLO's inbuild
eigenvalue solver.

If you want to setup the linkage yourself:
the MODULES/Makefile contains MKLLDFLAGS if MKL was requested by
answering yes as explained above. Find out your MKL installation and
edit this variable accordingly.

After all questions are answered there will be a lot of system specific output,
but you do not need to bother about it unless you suspect trouble.

Now, change into the FPLO14.00-45 directory. Start the compilation procedure and
install the executables in FPLO/bin:

  cd ..
  make
  make install

Under certain circumstances it can be necessary to clean up the source
tree first (e.g., if you compiled FPLO-14.00-45 previously on another architecture
accessing the same file system):

  make clean
  make
  make install

Now FPLO-14.00-45 is ready to work. To simplify its use, add the directory where
your executables reside to your PATH :

for sh users:

  PATH=$HOME/FPLO/bin:$PATH ; export PATH

for csh users:

  setenv PATH $HOME/FPLO/bin:$PATH
```

```
To make this permanent for the next login,
you have to add this line to your .profile or .login.csh respectively.
For bash .bashrc and for tcsh .tcshrc should be the correct places.
For local specifics you should ask your local system administrator.

Now you can read the Getting Started to learn how to perform calculations.

If you want to play with compiler options go to IV. Advanced Installation.

IIIa. Simple Installation on heterogenous Linux clusters
========================================================================
If you need several compilates for different machines on the same cluster
do the following.

Go into the directory FPLO14.00-45/install.

To create the appropriate Makefile for your architecture, type:

  sh ./MMakefile 1


enter the build-branch name, when asked (e.g. machine name ) and
answer the next question with yes if the executables shall have the branch name
appended at their end.

Now go one directory down and make the specific stuff
Note, that {branch-name} is a place holder for the actual name you
gave during the MMakefile invocation.

  cd ..
  make -f Makefile.{branch-name}
  make -f Makefile.{branch-name} install

This will make compilations in obj_{branch-name} sub directories.

When cleaning is needed use

  make  -f Makefile.{branch-name} clean
  make  -f Makefile.{branch-name}
  make  -f Makefile.{branch-name} install
```

```
IV. Advanced Installation
=========================
We tried to create a rather flexible install script. The script MMakefile
in FPLO14.00-45/install is controlled by a configuration script located in
FPLO14.00-45/install/conf. It is called either <CC>-<F90>-<OS>-[RELEASE] or
<CC>-<F90>-hostname.

     <CC>    is the name of the C-Compiler which you want to use
     <F90>   is the name of the F90-Compiler which you want to use
     <OS>    ist the name of the operating system on which
             MMakefile is running (you may get this name by typing uname -s)
     [RELEASE] is the (optional) release number of the operating system
             on which MMakefile is running (type uname -r)
     hostname is the hostname of the host on which MMakefile
              is running (type uname -n)

In these configuration files, architecture specifics are contained,
concerning pathes, libraries as well as similar things and compiler flags
needed to compile FPLO-14.00-45.

MMakefile is looking for a configuration file in the following sequence:

     <CC>-<F90>-hostname
     <CC>-<F90>-<OS>-[RELEASE]
     <CC>-<F90>-<OS>

Now we have to distinguish several cases:

  You use a combination of architecture and compilers which is tested and
  supported by us, but want to take other compile options: Go to
  1. Other compile options

  You use an architecture which is supported by us but you want to take
  other compilers: Go to 2. Other compiler

  You want to use an unsupported architecture: Go to 3. Other architecture

1. Other compile options
Let us assume you want to compile FPLO-14.00-45 on a 32bit-system running Linux
with ifort but options different from the default options.
The name of your host is, e.g., MyHost. Then you have to go
```

into the configuration directory and copy the appropriate configuration file gcc-ifort-Linux-i386 to gcc-ifort-MyHost. After that you can edit the lines concerning the compile options (CFLAGS, CFEDITFLAGS, F90FLAGS, SPECIALF90FLAGS), run MMakefile in FPLO14.00-45/install and compile in FPLO14.00-45 .

```
cd FPLO14.00-45/install/conf
cp gcc-ifort-Linux-i386 gcc-ifort-MyHost
vi cc-f90-MyHost
cd ..
sh ./MMakefile
cd ..
make clean
make
make install
```

2. Other compiler
Let us assume you want to compile FPLO-14.00-45 on a 32bit-system running Linux with with gcc but different F90-Compiler. The name of the new F90-Compiler is MyF90, your host is e.g. MyHost. Then you have to go into the configuration directory and copy the appropriate configuration file gcc-ifort-Linux-i386 to gcc-MyF90-MyHost. After that you can edit the lines concerning the compilers (F90, F90FLAGS, SPECIALF90FLAGS), run MMakefile in FPLO14.00-45/install and compile in FPLO14.00-45. Be careful with the compile options, at least you have to switch on ansi behaviour. Unfortunately, it is impossible to give more hints for choosing appropriate compile options.

```
cd FPLO14.00-45/install/conf
cp cc-f90-OSF1-V5.0 cc-MyF90-MyHost
vi cc-MyF90-MyHost
cd ..
sh ./MMakefile
Important:  Answer the question about F90-Compiler with MyF90 !
cd ..
make clean
make
make install
```

3. Other architecture
This is the most difficult case, and we can give you only some general hints:

Choose a configuration file that seems to stem from a similar architecture,

```
copy it on a file with the known naming conventions, than read every line
carefully. Some are self-explanatory, for others you will need hints about
your operating system.
Please read FPLO14.00-45/install/README.advanced for more
information. Feel free to contact us in difficult cases.


V. Miscellaneous
================
  You can change the names of the executables to be created in the
  following files:

  FPLO14.00-45/MODULES/Makefile.src  (fplo, dirac)
  FPLO14.00-45/BP/Makefile.src  (bandplot)
  FPLO14.00-45/FEDIT/Makefile.src  (fedit)

  You can change the directory where the executables reside in the
  following file:

  FPLO14.00-45/install/MMakefile  (change the line: installdir=$srcpath/bin )

  In the case you changed the executable's names and/or the install
  directory, please repeat the install procedure, in order to let the
  changes taking place.

For bugs in this document please send an email to fplo@ifw-dresden.de.
```

To install XFPLO and XFBP go into your-installation-path/FPLO/XFPLO_rel and your-installation-path/FPLO/XFBP_rel. Follow the instructions of the README in these directories.

1.2 Program structure

The FPLO package consists of the source tree, which you hopefully compiled succesfully, and of the executables, which reside in `$HOME/FPLO/bin` in the standard installation. See installation instructions in `FPLO.../install/README...` for more information. There are a couple of binaries and a number of scripts. The binaries are the following:

FPLO — The bandstructure solver for the Kohn-Sham problem of bulk systems and molecules. It is a monolithic program, which performs the whole self consistent calculation. There are no such things as a bundle of standalone programms, which handle subtasks of the whole problem. The input of FPLO is handled via the input editor.

FEDIT — The input editor. It handles the input editing for FPLO, DIRAC and BANDPLOT. There is basic help available via

```
FEDIT -h
```

Furthermore, there are help screens for every menu, which explain a number of aspects. **Please read them.** FEDIT has a pipe-mode, which is designed for automatic input management. It allows manipulation of input files by scripts, without messing up the input structure.

XFPLO — Display and manipulate structures, display fermi surfaces, define path through BZ for band plotting. Sec. 4.4

XFBP — (**X**11-**F**PLO-**B**and**P**lot) Plot bands and other functions, this program works a lot like xmgrace. Sec. 4.5

DMATEDIT — Edit LSDA+U occupation matrices for setting up starting configurations to enforce possible metastable solutions.4.6

FADDWEI — Add/manipulate weights in band weight files. Sec. 4.3

FOPTICS — Manipulate the optics output to generate other optics functions from `+imeps`. See Sec. 4.7

GREPFPLO — extract information from output files.4.2

BANDPLOT — **Old, since version 14 XFBP provides better services.** The bandstructure plotting utility. It is normally used by FEDIT (and not called from the command line) to create band-structure and band-weight plots. Call

```
FEDIT -bandplot
```

DIRAC — The standalone atom solver (spherical atoms). This solves the (non-) relativistic DFT problem for spherical atoms. To edit the input, call

```
FEDIT -atom
```

To learn more read the help screens in this mode. **(SIC is not yet working!)**

FPIOTEST — A utility for input manipulations. This is mainly used by the scripts of the distribution.

1.3 Input structure

The main input files use a special syntax, which has some aspects in common with the C-programming language. It is, however, not one of the common data languages around, but a private one. The reason to introduce this feature was the general design of the input handling and the aspired independence on external libraries, which can not be assumed to be present on every machine. The package contains an input parser, which is accessible to the C and F90 code. This reduces the need of FORTRAN IO management and thus increases the flexibility of version management and such things.

As a consequence, the user normally cannot and **should not** alter the content of the input files. **All input settings are meant to be changed with the help of** FEDIT **only.** (Exceptions are manipulations, which are done by XFPLO.) In fact FEDIT is very easy to handle and there is no need for manual input file manipulations. (Some scripts also change the input, using FPIOTEST, which in turn uses the mentioned parser to achieve this goal.) For batch jobs/scripting/automation there is a special mode called pipe-mode (Chap. 97) in FEDIT.

The input, which is needed by the various executables is managed in a particular way. The executables create communication files (+fedit, +fedithelp), which tell FEDIT how to process input and how to manage menus. FEDIT uses some methods of back-communication[2] to the executables to have them responding in certain ways. This kind of communication is designed to avoid the user to edit input files directly (with a few exceptions).

If FPLO is called in an empty directory, it will imediatly terminate with a message telling that it created one of those communication files. On the next invocation it creates standard input files and again terminates. All this is not seen by the user, if he uses FEDIT.

Important: To save a calculation it is sufficient to save all files of class 1 with the prefix '=.' (see Chapter 3). If one uses such an archived calculation later to, say, create some additional output data, the first call to FPLO would stop as described above (unless FEDIT was used before). So, dont be afraid if the code exits, stating that it created the communication files. Just restart it and everthing will be fine.

[2] These are command line options (BANDPLOT), status variables in the input files (=.sym) and deletion of certain files.

Chapter 2

Version control

The package uses rather strict version control rules. A version number has the form "`x.xx`", where `x` is a placeholder for a digit. Furthermore there is a release number, which has the form "`x`". A full version-release number is the version number followed by a "`-`" followed by the release number like in `14.00-45`, which means version 14.00 release 45. Additionally, a string is attached to the executable names in order to differentiate between different architectures, e.g. "-i386" or "-x86_64". Optionally, the user can chose to add a specific name at the end during the installation process, which is usefull on heterogenous clusters, where different compilations for different architectures are needed. This is achieved by adding the option "1" to the call: "`install/MMakefile 1`"during installation. Read the `install/README (Sec. 1.1)` file.

Every major input file contains its own version number. Every executable has its own internal version number. To avoid problems, one usually cannot use files with programs of different version.

There is a simple rule: **The version number of the code is changed whenever input has changed.** If only the release number has changed the input is not altered. Normally, changes of the input consist of adding something. In rare cases, the structure changes. For this reason it is not recommended to manipulate version numbers of files by hand.

2.1 Upgrading

If the version number of FPLO has been increased it is rather simple to upgrade the old files. FPLO upgrades the files in the working directory automatically if they have older version numbers. Just call FEDIT or FPLO and have a look at the output. If FEDIT is used it will ask the user before performing the upgrade, while FPLO will do it without asking!

Attention: We strongly recomment to copy the whole directory (assumed you organize different calculations in different directories) before executing the upgrade. The reason is that the numerical results between different versions may slightly differ due to numerical changes/improvements[1].

[1] After all FPLO (like any other code) solves the problem approximately, although rather accurate.

2.2 Downgrading

Sometimes, it is nessecary to downgrad files to an older version number, mostly to undo erroneous upgrading of files belonging to a series of calculations, which is intended to be completed with the older FPLO version. However, sometimes one may whish to recalulate something with an older version for comparison (although this should be a very rare case for the normal user).

Downgrading means that some information gets lost. Thus, you should consider to **save a backup copy** of the files **before** you **downgrade**. Have in mind that some calculation-modes are not available in older versions of FPLO. Downgrading of calculations using such modes makes no sense.

Downgrading is a bit more complicated than upgrading. There is a `perl` script in the distribution, which handles the complicated restructuring.

1. Execute `fdowngrad.pl` at the command line and answer a few questions. As a result the input files are downgraded. The old files are copied to a backup directory called `fdowngrad_backup[n]`, where `[n]` is a number which is increased on every call to the script.

2. Next call the older version of FEDIT belonging to the downgrad-version to regularize the files. Go to the `symmetry submenu` and use `update`. **NEVER SKIP THIS POINT!**

   Attention: Downgrading from version $\geq$ 9.07 to version $<$ 9.07 includes a subtle step, which will be explained here in detail. The density file information has been enriched in several versions. If one would use the file as it is with an older FPLO version, the density would be interpreted in the wrong way[a]. To avoid this it is nessecary to map the new density onto the old file format, which can only be done with an FPLO version belonging to the density file ($\geq$ `9.07`). The nessecary steps are

   1. Edit the file `=.densconvert`: put in the main version number (like 9 or 7).
   2. Run the FPLO corresponding to the density file. It will stop after the conversion.
   3. Delete `=.densconvert`, otherwise the program might get confused.

   Important: `fdowngrad.pl` tries to do these steps for you. This will only work if you have the proper FPLO version, idealy the one which has the version number of the files to be downgraded. If everything fails, you still have the backup. In the worst case you need to re-iterate the density[b].

   [a] This basically results in the destruction of the (already converged) density.
   [b] :-(

Pitfalls: Some information from files of newer version may be invalid in older version executables. This is seen if the older version FEDIT is used on the downgraded files (at least on exit, there will be a message.). Correct the input and rethink if you really did what you wanted to do.

Example:

Downgrading a full relativistic input say version `4.00` to version `3.00` (where full relativistic mode is not available) will leave the relativistic mode-data untouched in `=.in`. But, this is

invalid input in `fplo3.00-6`. Calling `fedit3.00-6` and executing `quit/save` will result in an error message about an invalid value of the variable `relativistic`. Go to the `relativisitc`-select box and select a proper option. Now, you can `quit/save` and you have valid input.

Chapter 3

Files

3.1 General

The files used by FPLO are classified into 3 major classes.

1. Files, which contain input or both input and output data and which are nesseccary for a succesful restart of a previously converged calculation. These have the prefix '=.'. One should not delete these files nor use them for different calculations with different parameter settings. A safe rule is: each calculation is done in a seperate directory. One may copy all these files into a new directory, modify the input with FEDIT and start a new calculation. This is especially usefull in the case of slight changes of some parameters, in which case the density of the previous calculation is usually a better starting point than the atomic density (created in the very beginning of a calculation if `=.dens` is absent). If, however, the number of sites in the unit cell or the type of elements is changing, an initial density is needed (see FPLO output).

2. Files, which are mainly output files and which are not nesseccary for a succesful restart of a previously converged calculation. These files have the prefix '+'. In general one can delete these files (`rm ./+*`) after successful calculations.

   Caution: the bandstructure is written into `+band...` or `+bweights...`. These files are used by XFBP (BANDPLOT) to create a postscript picture of the bandstructure. So, if one inadvertently deletes these files one needs a single step calculation, starting from a converged density, to recreate them.

   Important: Many (new) unix tools will interprete the '+' sign as an option flag. To use these tools with '+'-files, specify `./+file` instead `+file` on command line[1]. Example:

   ```
   less ./+band
   ```

[1] When FPLO was designed, this habit of unix (Linux) tools was not yet widespread, and hence not recognized by the FPLO authors.

3. Files, which are either pure output or status memory files for graphical tools. These files have no prefix.

   Example: `.net`-files, `.ini`-files

In the following the most important files are described in more detail.

Most input files are copied to the output. The script `fout2in` can be used to extract input files from any output file.

3.2 File class 1

The prefix '`=`' indicates their primary use as input to FPLO. Nevertheless, they are partially output files. FPLO uses the following files[2]:

`=.sym` and `=.in` are handeld by FEDIT , it is **no recommended** to manipulate them manually

---

`=.sym`

contains the crystal symmetry. From the information contained in this file `=.in` (and before version 6 `=.basis`) are recalculated. Recalculation happens if `=.in` is absent or if the status flag in `=.sym` requires an update. Normally this is done by FEDIT. On update, the non-default settings of the existing `=.in` will be retained unless the symmetry in `=.sym` contradicts these settings, in which case the default settings are used. The code will notify these reset-events after update[3].

In a standard FPLO calculation run (no update action) the symmetry settings of `=.in` are used even if `=.sym` contains a different symmetry. Normally, if the update function of FEDIT was used, the symmetry settings of both files are equivalent.

If `=.in` exists and `=.sym` is absent, FPLO will extract a valid `=.sym` from `=.in` (this is usually done while invoking FEDIT).

`=.in` contains the major control data for FPLO(including a copy of `=.sym`), except for the the basis. You can manipulate it interactively with FEDIT or automatically with the FEDIT-pipe-mode. Please read FEDIT help screens! Some information can be modified via XFPLO.

---

[2] There may be more class-1 files, which are not documented here. However, the normal user is not expected to need them. There are some utility programs for the package, which use the same file name convention and thus have additional class-1 files (e.g. BANDPLOT). These are also not documented here.

[3] When FEDIT is used, see the protocol screen after `symmtry update`.

`=.dens` contains the density contributions of all sites (in terms of radial functions, which are the coefficients of the angular momentum expansion of the **overlapping** site densities). This file serves as input and ouptut for FPLO. It is created by means of a simple atom-like calculation on FPLO startup, if not yet existing. The density file may be re-used in other calculations (often a better starting point than the atom densities) if the number of sites in the unit cell and the type of atoms are equivalent.

`=.dmat_init` contains the occupation number matrices for LSDA+U. This file's content is duplicated in `=.dens`, so `=.dmat_init` can get lost. However, if it is present it will overrule the information contained in `=.dens`. When FPLO runs it writes/updates this file with the current occupation number matrices. Its main purpose is to present the data in an easily editable format in order that one can manipulate the (initial) occupation number matrices to drive the calculation to one of the possible (local) LSDA+U minima. The format is quite unrestricted, however not fool-proof. So be careful when editing. In any case this file will be recreated in the next FPLO run. If you want a fresh copy just delete it. `New:` DMATEDIT can be used to manipulate this file in order to set up starting conditions for (meta) stable solutions. This program allows to rotate the local axis in which the orbitals are defined. (Sec. 4.6)

`=.kp` If this file is found by FPLO, the bandstructure (written to `+band...` (see Page 23)) is calculated at the points given in `=.kp`. This is e.g. used to calculate Fermi surfaces with XFPLO (Sec. 4.4.27). The points are given in units $\frac{2\pi}{a_0}$, thus i.e. for bcc lattices the line Gamma-H consists of all points between $(0,0,0)$ and $(1,0,0)$.

Format:

line 1: number-of-k-points [only partially occupied bands] [lower offset] [upper offset]

line 2,...: one k-point per line (3 real numbers, separated by space)

Explanations:

The three entries behind the number of k-points in the first line are optional. They are used by the newer versions of the program XFPLO to reduce file size.

[only partially occupied bands] a logical (`t/f`). If this is `t` only partially occupied bands are written to the files `+band...` or `+bweights...`.This reduces file sizes considerably, especially when used in conjunction with the Fermi surface program XFPLO .

[lower offset] Additionally that number of bands below the lowest partially occupied band are written to the files.

[upper offset] Additionally that number of bands above the highest partially occupied band are written to the files.

=.basdef The default basis is used if this file is not present. This file is copied to the output like the other main input files. Hence, we can extract the (default) basis definition file from any output. Use `fout2in -b` in order to extract the basdef, used in the output file under consideration.

=.unfold Unfolding is explained in a separate document (unfolddoc.pdf). Create the input by hand or use the unfold editor of the structure mode of XFPLO. Sec. 4.4.16

=.addwei Add band weights. Used by FADDWEI.(See. 4.3)

=.bwdef Molecular/individual band weigths can be defined via this file (Sec. 4.4.25). The advantage is that the resulting +bweights... file can be much smaller if the user only needs a few fatbands. Furthermore, molecular patterns help elucidate bonding behaviour. Call XFPLO `-bw` or XFPLO `=.bwdef` to edit the file. This file can be specified in the FEDIT bandplot menu after which FPLO will use the bandweight definitions from this file instead of the default when creating fatbands.

=.densmap Map density files from one structure onto another structure. See Sec. 4.4.28.

=.xstr Saved by XFPLO . It contains all settings needed to replicate the structure, lighting, fog, polyhedra and so forth. See Sec. 4.4.26. This has nothing to do with the structure used by FPLO.

=.xef Saved by XFPLO . It contains all settings needed to replicate the Fermi surface plot. See Sec. 4.4.27.

=.xfp Saved by XFBP . It contains all settings needed to replicate the plot created by this program. See Sec. 4.5.

`=.cmd` Saved by XFBP . It contains a user written script used by this program. This enables automation. See Sec. 4.5.

`=.coeff` **Obsolete since version 10.00,** because it has benn turned into the switch `Ouput +coeff file` in the FEDIT bandstructure submenu.

`=.atcharge` **Obsolete since version 10.00**, because input was moved to FEDIT charges menu. On version update the data is transferred to =.in (FEDIT).

Somebody may want to use the virtual crystal approximation (VCA), which consists basically of introducing non-integer nuclear charges. This may be done by defining the content of this file.

Format:

line 1: number of lines following

line 2,...: number of Wyckoff-position followed by the nuclear charge

The nuclear charge must not deviate from the nominal nuclear charge Z of the element by more than ± 1. Since basis sets are element specific it might make a difference, whether one uses $Z-\delta$ or $Z+\delta$. The basis set is always chosen according to the nominal nuclear charge. However, basis parameters (and also other specific paramters) are interpolated between Z and $Z \pm \delta$.

`=.mol_charge` **Obsolete since version 10.00**, because input was moved to FEDIT charges menu. On version update the data is transferred to =.in (FEDIT).

In molecule mode this defines the allover cluster/molecul charge. The file contains one real number (the charge) in the first line. A positive number means less electrons than neutral.

`=.basis` (obsolete since version 6.00) contains the basis definitions. This includes the definition, which orbitals enter the calculation (core/ semi-core/ valence/ polarization) and the inital definition of the compression parameters (x_0) of the confining potential. The file content may be manipulated using FEDIT.

In auto optimizing mode (option `BASIS_OPTIMIZATION`) the file is updated with the new value of x_0 while FPLO is running. So, be aware if FPLO is running and FEDIT is used meanwhile,

that the file content may have changed on disk during the FEDIT session. You will be promted to overwrite `=.basis` on exit (`quit/save`) in case that the content changed, while the FEDIT session was active. Usually it is correct **not** to overwrite the `=.basis` content in such a situation.

For all these files, read the help screens of FEDIT!

`=.densgrid` **Obsolete since version 9.00**, because input has been moved to a new FEDIT `sub-menu.`

To create real space density or potential plots create this file. It contains header informations and grid data. It may contain comments (line starting with '`#`') everywhere. It may contain empty lines.

Comments at the top of the file (before data lines) may contain control settings. All other lines contain real space vectors in cartesian coordinates. For each vector the density/potential is plotted into the file `+densgrid`.

Header:

Important: please **no** tab-characters, only space!
All header control data are optional. Below we give the complete list of possible controls and their options. Any combination and order of options is valid. Options must be separated by at least a single space. The colon after the control key-word (`output`,`data`,...) may be separated with spaces. Any of the control lines described in 1-3 may be omitted, which forces default behaviour.

1. `# output: comments emptylines emptylines_with_space stop`

`comments`	output all lines starting with any space followed by '`#`'. If '`# output`' is the first line, it controls the output of all lines following, including itself.
`emptylines`	copy all empty lines of `=.densgrid` into `+densgrid`
`emptylines_with_space`	put a line with one single space into `+densgrid` for every empty line of `=.densgrid` (some plotting programs need this to separate data sets)
`stop`	stop after plotting, please be aware that you need a converged calculation to get reasonable densities/potentials!

   `emptylines_with_space` wins over `emptylines`, if both are specified!

2. `# data : index point total spin spinup spindown`

`index`	print index of grid point in first line
`point`	print grid point

Table of Contents | Index

`total` print total density/potential

`spin` print spin density (i.e. $n^{\uparrow} - n^{\downarrow}$)/xc-field

`spinup` print majority density/potential

`spindown` print minority density/potential

3. `# type : density potential`

   `density` plot density data

   `potential` plot total-potential data

The last `type` specified, will win.

Plotting is done at the beginning of each iteration cycle after the potential is calculated.

Default:

1. `type:` density
2. `output:` no comments, no empty lines no stop
3. `data:` point total spin spinup spindown

The order of output data is fixed. So the `data`-options may be given in any order, but the output in `+densgrid` allways has the order: index point total spin spinup spindown. Of course, some of the fields may be absent, if omitted in '`data: ...`'

Remark: If some grid point falls onto an atomic position, the onsite contribution of this atom is neglected, since relativistic densities diverge at the nucleus ($s^{1/2}$ and $p^{1/2}$ orbitals diverge). Same holds for the potential (relativistic or not).

`=.ldos` **Obsolete since version 4.01**, because input was moved to FEDIT `bandplot menu`

To create lm-resolved densities of states, create this file in the directory, where the calculation is done. If FPLO finds this file the local orbital-resolved DOS is calculated.

Format:

line 1: number of lines following

line 2,...: number of site, for which a resolved DOS should be created

'sites' means all sites in the cell, not the Wyckoff positions! The definition of sites is found in the FPLO-output section '`UNIT CELL CREATION`', part '`Atom sites`'.

3.3 File class 2

The prefix '+' indicates the primary use as output from FPLO. Nevertheless, the files are partially input files for subsequent runs or tests. (Examples: debug files like +basis, +loi)

+dos.total This file and all other DOS files (except the +(i)ldos files) are created if the option "CALC_DOS" is set or if "Bandstructure plot" in the bandplot menu is true unless the option "NO_DOS" is set.[4] It contains the total density of states.

+dos.total.l contains the l-projection of the total density of states. E.g. the d-DOS is the sum of all d-orbital contributions of all atoms to the total DOS. See comments inside the files to see which angular momentum l is contained in the file. The numbers in the file suffix are just counters.

+dos.sort contains the sort projected DOS. This is the sum of all DOS contributions of all sites belonging to the same Wyckoff position (sort).

+dos.sort.nl contains the sort and nl-projected DOS. This is the sum of all nl-DOS contributions of all sites belonging to the same Wyckoff position. See comments (lines starting with '#') in the file.

+idos... These are like the +dos.-files but with the integrated DOS. Integrated DOS is only created when the option Plot IDOS in the bandplot menu is set.

+(i)ldos.site.nl These files are created for all sites given in the bandplot menu entry 'Local DOS sites' (this entry is a space separated list of site numbers) (see page 21 for older input versions) and contain the site and l,m-resolved DOS. The file numbers nl... are running indices, the real nlm numbers are written in comment lines ('#') inside the files. In full relativistic mode also $lj\mu$ projected files get produced.

+imeps This file contains the inter-band part of the optical function $\mathrm{Im}\varepsilon(\omega)$ it is used for optics (see Sec. 89).

---

[4] In CPA calculations there is no "bandstructure plot" option. Instead the Bloch spectral density may be calculated.

+coeff If coeff output is switched on in the FEDIT bandstructure submenu this file is created and contains the wave function coefficients C of the reduced valence problem

$$HC = SCE$$

This file is usually very large. It can however be used in conjuntion with XFPLO -bw to create hand tailored band weights.

Attention: The full wave function $\Psi = \Phi C$ is constructed from this information (C) **and** from the core and valence orbitals $\Phi_{c,v}$ and core-valence overlap S_{cv}, which are not contained in +coeff.

+error Created (and updated), while FPLO is running. It contains a summary of warnings and error messages. The update mechanism does not work on some platforms. Thus at the moment, it is best practice to check the FPLO output, since all messages are duplicated to standard output.

A lot of messages contain the number of the iteration step, where they occured. Normally, only the messages of the last iteration step (before convergence) are relevant.

+run contains the hostname and the PID of the last run of FPLO. If it is still running, this information may be used to kill the job.

Caution: Killing FPLO, may cause loss of the =.dens file and therefore loss of the calculation result, in case that FPLO is just writing the file =.dens, when it is killed! (The same holds for =.basis.)

+band..., +bweights... are created if the band-structure/band-weights plotting options have been set via FEDIT. They contain the band-structure/band-weights. Use XFBP filename (or before version 14 FEDIT -bandplot[5]) to produce the related pictures from them. Since version 14, a suffix is appended to +b... files (Table 3.1). Also since version 14, two band weight files are produced in full-relativistic mode, one with $jl\mu$-orbital projection and one with (approximate) $lm\sigma$-projection (see Sec. 4.3).When unfolding is active, both default and unfolded files will be created.

[5] This will use BANDPLOT to create a Postscript file.

cases	not full-relativistic	full-relativistic
default	+band +bweights	+band ljm +bweights $lm\sigma$ +bweightslms
=.kp meshes (Fermi surface)	+band_kp +bweights_kp	+band_kp ljm +bweights_kp $lm\sigma$ +bweightslms_kp
unfolding	+band +bweights +bweights_unfold	+band ljm +bweights +bweights_unfold $lm\sigma$ +bweightslms +bweightslms_unfold
=.kp and unfolding	+band_kp +bweights_kp +bweights_kp_unfold	+band_kp ljm +bweights_kp +bweights_kp_unfold $lm\sigma$ +bweightslms_kp +bweightslms_kp_unfold

Table 3.1: Band weights files.

If the XFPLO =.bwdef mechanism (Sec. 4.4.25) is used the resulting weights file is named by the user.

It is always a good option to check which files got created after a run (`ls -ltr`).

+points Created in the initialization phase at the beginning of the FPLO run. It contains the special symmetry points used for the band structure creation. It is used by XFBP (old: BANDPLOT).

+symmetry Created by the symmetry module of FPLO. It contains information about the crystal symmetry.

+fcor.sort.spin These files are created, if option PLOT_BASIS is switched on. They contain the radial part of the core orbitals.

`+fval.sort.spin` These files are created, if option `PLOT_BASIS` is switched on. They contain the radial part of the valence orbitals. Here valence orbitals include all non-core orbitals.

`+fkval.sort.spin` **and** `+fkcor.sort.spin` The kinetic core and valence functions. This is supplied mainly for debugging purposes.

`+atpot.sort.ivat` (**New since version 7.00**) Created if option `PLOT_REALFUNC` is switched on. Contains the effective potential, used in the Hamiltonian of the radial atom like equation for the calculation of the basis orbitals

$$V = \frac{l\,(l+1)}{2r^2} + \langle V \rangle_{\text{spherical}}$$

$\langle V \rangle_{\text{spherical}}$ is a spherical potential defined by the basis parameters. If two basis orbitals have the same parameters, they belong to the same class and have the same atomic potential. The index `ivat...` labels the different classes. These files are supplied for debugging purpose.

`+dens.site.spin`, `+har.site.spin` Created if option `PLOT_REALFUNC` is switched on. All these files contain the angular momentum components. They contain

- The density contributions of site 'site'.
  The $L = 0$ part is multiplied with $\sqrt{4\pi}r^2$. So, it integrates to the electron number belonging to the respective site.
- The local multipole-neutral Hartree potential contributions of site 'site'.

Be aware, that the local contributions have no physical meaning, only the sum of all has one. These files are supplied for debugging purpose.

`+fedit`, `+fedithelp` Communication files between the executables FPLO/ BANDPLOT/ DIRAC and FEDIT. These files are created on every run of the executables. In the initialization sequence of the FEDIT run, they are deleted and the appropriate executable is called, to recreate them. In this way it is assured, that FEDIT allways reads the correct information.

`+grid_dens.***` These files are created by the new grid-output module and are explained in the grid-output sub-menu.

Table of Contents | Index

`+plasmon` This file contains the main axis and energies of the plasmon tensor.

`+voronoi` (**Obsolete since version 6.00**) Created in the initialization phase at the beginning of the FPLO run. Contains the voronoi cell geometry.

If '`build voronoi`' is true, `+voronoi` is created and used. If '`build voronoi`' is false, one may specify '`voronoi file`' as an alternative file to read from. This serves for cell merging. However, cell merging is not expected to be needed in versions later than `3.00`.

`+symanalysis` (**Discontinued since version 6.00**) Created by the symmetry module of FPLO. It contains information about the crystal symmetry induced conditions for the (non relativistic) matrix elements of the onsite blocks of the density-matrix/Hamiltonian/overlap-matrix. The diagonal elements give the conditions for the site and angular momentum resolved DOS.

`+fdval.sort.spin` (**Dicontinued since version 6.00**) These files are created, if option `PLOT_BASIS` is switched on. They contain the radial part of the derivative of the valence states with respect to the prefactor (λ) of the confining potential

$$V^{\mathrm{cf}} = \lambda \left(\left(\frac{r_{\mathrm{NN}}}{2} \right)^{\frac{3}{2}} \right)^{-4} r^4$$

$$\lambda = x_0^{-4}$$

`+atcor.sort.spin, +atval.sort.spin` (**Dicontinued since version 6.00**) Created if option `PLOT_REALFUNC` is switched on. Contains the effective potential, used in the Hamiltonian of the radial atom like equation for the calculation of the basis orbitals

$$V = \frac{l\,(l+1)}{2r^2} + \left\langle \left\langle V^{\mathrm{cryst.}} \right\rangle_{\mathrm{spherical}} \right\rangle_{\mathrm{smoothed}}$$

$\langle V^{\mathrm{cryst.}} \rangle_{\mathrm{spherical}}$ is the spherical average of the crystal potential around the considered site. The potential actually used (and contained in the files) is a smoothed version of V!

`+dirsh` (**Obsolete since version 6.00**) Created, if shape test is performed. It contains the shape function along the lines specified in the shape sub menu of FEDIT.

`+unity` (**Obsolete since version 6.00**) Created, if shape test is performed. It contains the sum of all shape functions minus 1 along the lines specified in the shape sub menu of FEDIT.

It should contain zeros, at least near the origin. (For larger distances the summation of all shape functions may be incomplete.)

`+densgrid` See `=.densgrid`

3.4 File class 3

These files have no prefix.

`dmatedit.ini` DMATEDIT saves the local axes and stuff in this file.4.6

`grid_dens***.[net|general|cfg]` These files can be created by the new grid-output module. They define the gridded data structure and an `opendx` program to display it in opendx. This is explained in the grid sub-menu.

`bravais.ps`, `primitive.ps` Created in the initial part of the FPLO run, before the density is read. Contains a postscript picture of the bravais/primitive cell.

`vcell{site}.ps` (**Obsolete since version 6.00**) Created in the initial part of the FPLO run, before the density is read. Contains a postscript picture of the voronoi cell of site 'site'.

3.5 Directories

FEDIT creates/uses a subdirectory `+tmp` in the directory where it was called, to perform input file updates. This directory may be deleted after use of FEDIT (or at the end of the calculations) In newer verions FEDIT tries to delete it itself.[6].

[6] Sombody will argue that we should use the systems `tmp` directory. May be. But our way of doing it is independend of the system settings.

Chapter 4

Programs

Some of the programs are explained here.

4.1 LSDA+U

The new basis since version 7 sometimes requires the gross population projector, since the new basis funcions are more extended than the FPLO<5 basis functions. Due to historical reasons the default is still orthogonal projection (net population). Please consider the gross projection, especially if your calculations do converge badly, especially for small volumes and cases of considerable ligand hybridization. Check the population analysis for overly large net occupations (much larger than gross). This might be a hint to use the gross-projector.

Now, full relativistic LSDA+U is implemented. Note, that this is less stable than non-full-relativistic LSDA+U. Main reasons are

- bands now carry non-integer spin weight, which allows reshuffling between spin occupations depending on tiny band shifts at the Fermi level
- symmetry is lower → more on-zero occupation number matrix elements
- at the same time now orbital momentum is a meaningfull shell property, which increases the number of solutions, between which to fluctuate. Furthermore, it seems to converge slower than the spin moment and total occupation.

In order to see the proper orbital momenta we have to project onto the spin quantization axis, since otherwise we can not compare S_z which is in the field coordinate system and L_z, which would be in the global coordinate system. This means that when using DMATEDIT you have to go into the proper system (axis dialog) to manipualte the orbital moment.

Note, that in non-spin polarized full relativistic LSDA+U time reversal symmetry is enforced.

4.2 Grep data from output: GREPFPLO

4.2.1 Specific data

The old `grEE` flavours of output greping have been enriched/replaced by the program GREPFPLO.

Usually different calculations are performed in different directories, with input values changing from calculation to calculations. If the directory names are given such that they contain the value in the name, output greping is straight forward. Let's suppose we have a bunch of dirs for varying FSM momenta and the dirs are called `M=1`, `M=2`, `M=2.3` and `M=2.5`. The output files are called out in all dirs. We can now create a data file containing the FSM momenta and total energies of all four dirs by calling

```
grepfplo -m EE -p M=
```

which results in a file looking something like:

```
1   -141.12
2   -141.13
2.3 -141.14
2.5 -141.13
```

Note, that we changed the table for the onsite orbital momentum to show entries for each site not each sort. This is repetitive information, but more consistent with other tables.

GREPFPLO has a help screen (option `-h`), which is shown below:

```
Extract data from the fplo output file(s)

Usage: grepfplo [-h] -m mode [options] (-p prefix) (-f outfile) ([-a(ll)])
          ([-x(fbp)])  ([-xm(grace)])

We assume that separate calculations are done in separate directories
and that the directory names contain the running variable value.

prefix: is the prefix string of a bunch of directories, of
which the data shall be extracted. (Default is . =current directory.)
The prefix is used to select the directories and is removed from the
directory name to get the corresponding values;
E.g. if the directories are named
M=0.1 M=0.2 M=0.3, the prefix M= will result in a data set
0.1 data1
0.2 data2
0.3 data3
 where data... is the data actually extracted from the output files. (See below.)
```

```
If the directories are named id=1_M=0.1 id=2_M=0.2 ..., the prefix
'id*M=' (quotes are important) will give the same list as above.
The prefix is used to identify the directories but also
is removed from the directory name to obtain the values of the
first column (which are encoded in the dir-names).

Use single quotes to protect * in shell context;

If prefix is . (simple dot), the file in the current directory is scanned.
In this case the option '-all' will make a list of the data indexed by the
iteration step.

outfile: is the name of the output file, which should be the same in all
directories. Default is out.


modes: Default is EE.
EE: extract the total energy. There is no options. Examples:

    grepfplo -m EE -p 'id*M=' -f out

          This extracts all Etots from all directories named id*M=...
          where * can be anything and ... after M= is usually the value of M for
          this particular directory/calculation
          The result is a table of M and Etot values

    grepfplo -m EE -p M= -f out

          Same as above, but for directories called M=...

    grepfplo -m EE -p . -f out

          Extract last total energy from file out in current directory
          Equivalent to grepfplo -m EE or grepfplo

    grepfplo -m EE -p . -f out -a

          Extract total energies for all iteration steps from current directory
          The resulting table has columns 'iteration step' 'total energy'
          Equivalent to grepfplo -a

    grepfplo -m EE -p . -f out -a -x
    grepfplo -m EE -p . -f out -a -xm

          Extract total energies for all iteration steps from current directory
          and send them to xfbp (-x) or  xmgrace (-xm)
```

```
          This is equivalent to grepfplo -a -x or grepfplo -a -xm


SS: total gross spin. no options.

it: iteration progress

fit: force iteration progress

time: timing of one cycle

term: termination of process

SSat: total gross spin of atom. Options: site-number. Examples
      grepfplo -m SSat 13 -p M= -f out

N_net/N_gros/S_net/S_gros:  individual population numbers.
      Options:  site-number orbital-number.
      orbital-number is the number of the orbital in the order as printed in the
      population analysis.
      If orbital-number is out of range the total site population number is printed.
      For the N_gros cases there is one more number than for N_net and S_..., which
      is the number of excess electrons of the site.
Lzat: the orbital moment of an atom. Options: site-number. Example
      grepfplo -m Lzat 12 -p U=

Bfsm: the auxillary magnetic field needed to set the FSM moment.

dBfsm:the change of the auxillary magnetic field needed to set the FSM moment.
      This is for controlling what's going on in full relativistic FSM calculations.
```

4.2.2 Categorized data

Several modules write their output with a prefix in order to easily extract the data. E.g. the topological insulator module prefixes all output with "`TI:`". In order to grep it and to pipe it to a pager or save it to a file you can do something like

```
grep TI: out | less
```

or

```
grep TI: out | more
```

or

```
grep TI: out | tee tidata
```

or

```
grep TI: out > tidata
```

Possible prefixes are listed in Table 4.1.

Module	Prefix	Comments
Forces	`FORCES:`	
Topological Insulator	`TI:`	
Optics	`OPTICS:`	
Molecular fatbands	`FATBANDS:`	
Normal iteration information	`SCF:`	
Force iteration information	`FORCES:`	
Preparing basis potentials	`vatom[Wyckoff number]` (e.g. vatom1)	Only shown, if verbosity level ≥ 4
Start density preparation	`startatom[Wyckoff number]` (e.g. startatom1)	

Table 4.1: Module output prefixes

4.3 Adding band weights: FADDWEI

Band weights are created if the `bandstructure plot` and `weights` options are set in FEDIT. This will produce `+bweights`. The bandplto submenu of FEDIT contains the option to define a coordinate system for the orbitals to project on (transform axes). In full relativistic mode additonally a projection onto quasi non-relativistic symmetries is provided in the file `+bweightslms`.

The full relativistic basis has quantum numbers ljm_j, while the non-relativistic basis has $lm\sigma$ (σ is spin up or down). In not full relativistic mode the spin is encoded by having spin up and spin down bands. This means that for non-spin polarized calculations only one set of bands/weights is needed. In full relativistic mode spin is not a good quantum number, hence a projection into $lm\sigma$ space is approximate. Furthermore, the spin is no longer distinguishing bands. Instead spin becomes a weight itself, which means that the orbitals are now carrying an explicit spin label. For non-spin polarized full relativistic calculations the resulting fatbands are written for both spin directions, for the reason that in systems without inversion symmetry there can be spin polarization of individual eigen functions even though the sum over all eigen functions always produces a non-polarized density.

The weights are normalized such that the sum of the weights of all orbitals at a certain point of the bandstructure add up to one. In full relativistic mode with $lm\sigma$ projection they add up to one summing over the spins, which are now part of the orbitals. If there is an inversion

center and a non-spin polarized calculation, Kramers degeneracy will lead to every band being doubly degenerate. In this case the spin-up an spin-down weights will be equal. A pure band with pure orbital character, say Fe $3d_{z^2}$ will actually have weight 0.5 for each spin, so that the weights appear half as small as compared to a not full-relativistic calculation. On the other hand there will be two degenerate bands (inversion center → Kramer) which effectivly gives the same counting as in not full-relativistic treatment. This has to be kept in mind when interpreting (adding) band weights in such cases.

As an alternative to the default route of producing fatbands the `=.bwdef` mechanism Sec. 4.4.25 can be used to create custom tailored weights.

Once a file `+bweights...` or another file containing band weights exists it is sometimes desirable to add several weights together to create coarser representations. E.g. the default weights for a $3d$ transition metal would contain weights for each individual orbital $3d_{xy}$, $3d_{yz}$, $3d_{z^2}$, $3d_{xz}$ and $3d_{x^2-y^2}$. The weights file contains a header, which names the character of each weight. In a default file these labels are named e.g. "Fe(003)3d-1", which stands for iron site 3 orbital $3d_{-1}$. In not-full-relativistic calculations the angular part of the orbitals are defined as real spherical harmonics Y_{lm} with respect to the global cartesian coordinate system. In this system we get the labeling Table 4.2.

In a full relativistic calculations the angular parts of orbitals are defined as spherical spinors.

$$\chi_{\kappa\mu} = \chi_{lj\mu} = \sum_{s=-1}^{1} c^s_{\kappa\mu}\chi_s \mathcal{Y}_{l,\mu-\frac{s}{2}}$$

with Clebsch-Gordon coefficients $c^s_{\kappa\mu}$ the complex spherical harmonics $\mathcal{Y}_{lm}$ and the spin-orbit quantum number κ: $\kappa\left(l, j = l - \frac{1}{2}\right) = l$, $\kappa\left(l, j = l + \frac{1}{2}\right) = -l - 1$. These spinors are completely specified by the quantum numbers l, $j = l \pm \frac{1}{2}$ and $\mu = -j, \ldots, j$. The indices $lj\mu$ span a space of $2\,(2l+1)$ orbitals, which is the the same number as for the non-relativistic lm orbitals if spin is counted ($lm\sigma$).

	real Y_{lm}				complex $\mathcal{Y}_{lm}$			
m	s	p	d	f	s	p	d	f
-3				$y\left(3x^2-y^2\right)$				$e^{-i3\varphi}$
-2			xy	xyz			$\mathrm{e}^{-i2\varphi}$	$z\mathrm{e}^{-i2\varphi}$
-1		y	yz	$y\left(5z^2-1\right)$		$\mathrm{e}^{-i\varphi}$	$z\mathrm{e}^{-i\varphi}$	$\left(5z^2-1\right)\mathrm{e}^{-i\varphi}$
+0	1	z	$3z^2-1$	$z\left(5z^2-3\right)$	1	z	$3z^2-1$	$z\left(5z^2-3\right)$
+1		x	xz	$x\left(5z^2-1\right)$		$e^{i\varphi}$	$ze^{i\varphi}$	$\left(5z^2-1\right)e^{i\varphi}$
+2			x^2-y^2	$\left(x^2-y^2\right)z$			$e^{i2\varphi}$	$ze^{i2\varphi}$
+3				$x\left(x^2-3y^2\right)$				$e^{i3\varphi}$

Table 4.2: Mapping spherial harmonics to functions. (**Normalization/Phases not included!**)

In a user manipulated file these labels can contain anything. They are however restricted to 17 characters. Now, it could be good to show fat bands for the whole $3d$ set of orbitals, which

means that we have to add $w_{\rm sum} = \sum_{m=-2}^{2} w_{3d_m}$. More generally, one could add all weights corresponding to a certain site or several sites and so on. This is achieved by FADDWEI. It supercedes the old programs `addweig` and `addweights`. The program has commentline flags (try `-h`).

FADDWEI reads one weight file (`+bweights...` or such) and one state-definition file. The default is `=.addwei`. When starting fresh it offers to create an example state-definition file. This file tells which weights/orbitals have to be added into a new weight. There are two options to do that. Either specify a complete label for each weight to be added, or specify an element a number of sites and a number of orbitals. Optionally, a factor can be given, which is multiplied to each weight of the corresponding specification before adding all up. This is most likely seldomly nessecary.

If pseudo-nonrelativistic projections onto Y_{lm} symmetries are created in full relativistic mode $lm\sigma$-projection or using the `=.bwdef` mechanism (Sec. 4.4.25) the default orbital labels contain an additional spin indicator (up/dn) at the end, in which case the input in `=.addwei` must specify the spin. In ordinary calculations the spin is encoded by haveing one band for each spin direction. This however is not possible in full-relativistic mode, hence the explicit spin in the default labels.

To obtain a complete (long) list of all labels use FADDWEI `-p`.

Examples:

- We have Fe at sites 2,3,4 and 8, oxygen at sites 12,13,14,15,16 and some other stuff. We define one weight with name Fe3d one with "O2p" and one with "Fe O all".

```
# all Fe 3d
name ''Fe 3d''
    atom Fe sites 2..4 , 8 orbitals 3d # all Fe 3d orbitals for sites 2,3,4 and 8
# all O 2p
name='O 2p'
    atom= O sites =12..16 orbitals= 2p # all O 2p orbitals for sites 12 through 16
# one weight sum containing Fe and O
name 'Fe O all'
    atom Fe sites 2..4 8 orbitals 3d
    atom O sites 12..16 orbitals 2p
```

- We have a weight file with labels "all Fe" "all O" and "K s" created by some other tools. Lets create one weight with them all summed up but with the "all Fe" weight being multiplied with factor 2

```
name all
    labels 'all Fe' fac 2
    labels 'all O' 'K s'
```

- We have a full-relativistic calculation and created pseudo non-relativistic Y_{lm} projections. Weights of Fe 3d are obtained via

```
name Fe_up
    atom Fe sites whateversites orbitals 3d spin up
name Fe_dn
    atom Fe sites whateversites orbitals 3d spin dn
name Fe
    atom Fe sites whateversites orbitals 3d spin both
```

- We want all orbitals of Fe site 2. (This would add all orbitals whose labels start with "Fe(002)" including all spins for pseudo-nonrelativistic weigths. If you want the spins separately add the orbitals explicitly)

```
# all weights whose label reads Fe(002)....
name 'All Fe'
    atom Fe sites 2 orbitals all
# all essential spin dn Fe orbitals from pseudo-nonrelativistic projections
name 'all Fe dn'
    atom Fe sites 2 orbitals 3s 4s 3d 4d 4p spin dn
```

The way the program works is to assemble all labels "El(site)orb[spin]" which can be created from the definition and sum them with optional factors. The resulting labels must exist int he file header. If the "`labels`" keyword is used the file header must contain these labels.

Quotes (" or ') can be used to include spaces into names and labels, but they are not needed when no space is contained in a name or label. Comments start with # and end at the line end. Commas can be used to separate list elements but are not needed. An = sign can be added after keywords for convenience (`name=Fe` instead of `name Fe`). The range specifier `2..6` is usefull and expands to 2,3, ... 6 for long integer lists.

There is the option (`ewindow`) to define an energy window, which when defined restricts the band written to the output file to bands, which are not completely outside this window. This saves time and makes the resulting files smaller.

The program writes the labels it selected to stdout. Please check the list.

Formal grammar:

- `#comments`

  can appear everywhere

- `weightinfile` file_name_of_input_weight_file

  this will define, which weight file (e.g. file+bweights...) is used as input. If the commandline option -f is used the command line argument file name will be used instead.

- `weightoutfile` file_name_of_resulting_weight_file

  this will define, which weight file (e.g. file+bwsum...) is used as output.If the commandline option -o is used the command line argument file name will be used instead.

- ewindow emin emax

  remove all bands, which are completely outside this energy window (in eV). This makes the resulting file smaller.

- `name` somename

  starts a new output weight adding all the weights defined by `labels`- or `atom`- keywords after this until the next `name`-keyword or end of file is met.

- `labels` list_of_input_weight_files_labels [`fac` number]

  extract and sum all weights corresponding to the labels given in the list. These labels must appear in the input weight file header. Optionally a factor can be specified, which is multiplied to each resulting input weight before adding them all up.

- `atom` element `sites` list_of_int_or_ranges `orbitals` list_of_orbitalnames [`spin` `up`|`dn`|`down`|`both`] [`fac` number]

  select all input weights with labels El(site)orb[spin] multiply them with the optional factor and add them to the output weight defined in the previous `name` clause. The site list can be a list of integers are ranges (e.g. `4..8`). Orbital names can be

nl:	selects all orbitals nlm. Example 3d: results in 3d-2, 3d-1, 3d+0, 3d+1 and 3d+2.
nlm:	select orbital nlm. Example 3d+0: results in 3d+0 (the plus sign matters!)
nlj:	select all orbitals $nlj\mu$: Example 3d3/2: results in 3d3/2-3/2, 3d3/2-1/2, 3d3/2+1/2 and 3d3/2+3/2
nljm:	select orbital $nlj\mu$: Example 3d5/2+3/2: results in 3d5/2+3/2
all:	select all labels, which are specified by the `atom` and `sites` keywords resulting in labels El(site)...

  Note, that the orbtial name nl will select the non-relativistic names. If you want to add the whole 3d-shell in a full-relativistic case use "3d3/2 3d5/2" to get the whole nl-shell.

4.4 Graphical interface: XFPLO

There are help screens, which explain basic operations in most places.

4.4.1 Introduction

Many controls show tool tips, when the mouse hovers over them for a short time. The help screens try to tell you what's important. However, some basis knowledge about the corresponding topic is assumed. The xfplo tool kit is under constant development and hence not always complete or consistent.

4.4.2 Trouble shooting

If you use the program remotly via ssh login and the central widget is not displaying anything (no white screen) try

```
export LIBGL_ALWAYS_INDIRECT=yes
```

an the command line before running the program.

4.4.3 Hotkeys

In the main window several hotkeys are defiend.

Mac users Ctrl is the Apple-key on Mac OS. Alt keys do not have an equivalent on Mac. Especially, there are no hotkeys to select menu entries or controls in dialogs. That's Mac OS's restriction. On Linux all these hotkeys are working.

Axes

Boundaries/Display cells

Table 4.3: Axes hotkeys

Action	Key
Show axes popup	a
toggle cartesian	a-x
toggle conentional	a-c
toggle primitive	a-p

Hotkeys

Table 4.4: Boundary/Display cell hotkeys

Action	Key
Show boundary popup	b
toggle main boundary	b-b
toggle conentional	b-c
toggle primitive	b-p

Hotkeys

4.4.4 The 3D View

Tool-buttons are used in the GUI. They have a main part, which acts like a button and a little down-arrow part, which acts like a combobox. When choosing from the combobox menu the corresponding action will be executed and the corresponding action becomes the buttons default action. So, pushing the button will execute the recently chosen action.

Moving in the sceene The 3D view has extensive mouse input actions.

Trackball rotate Hold down the left mouse button and move it around. The rotation is not commutative. So after a while one gets a pretty good feeling for orienting the sceene.

Circle rotate Ctrl+left mouse button 2D-rotates the sceene around the center of the scene.

Pan/Move middle mouse button or Shift +left mouse button will move the scene

Zoom the mouse wheele or the right mouse button or Ctrl+Shift+left mouse button will zoom the sceene in and out.

Default views Ctrl+1 — Ctrl+4 will set a default view angle/rotation around the object center. After moving the sceene center does not coincide with the object center.

Default pan Ctrl+R will reset the pan and zoom such that the object basically fits into the viewport. This is not allways prefect, though.

Canvas size The canvas or view has by default a fixed size. This comes in handy if several similar pictures have to be created. The canvas size can be set with a menu (Ctrl+V) or by unlocking the canvas by clicking the lock icon in the lower left corner of the window. Then window resizing will resize the canvas as well. One can lock the canvas at any size.

4.4.5 Atom distances and angles

Menu: Tools→Measure or hotkey M will open the measureing window.

Use the mouse to click on a visible part of an atom. The atom gets added to the list in the dock-window if it is not yet selected, otherwise it gets removed from the list. Atoms get added to the bottom of the list but can be removed arbitrarily. This will change the order of the remaining atoms and can be convinient.

Up to 3 atoms can be selected. If there are 3 atoms they form a sequence (1st,2nd,3rd). The distance between the 1st and the 2nd and the 2nd and the 3rd is shown in the dock-window, as well as the angle formed by the vectors (1st-2nd) and (3rd-2nd).

If M is pressed again the dock-window closes the measuring stops and the list gets reset.

If ESC is pressed the list is reset.

If the underlying data change (symmetry, boundary) the list gets reset.

4.4.6 Symmetry Dialog

4.4.6.1 Groups

The groups can be specified as space groups and points groups, depending on the structure type.

The point groups are specified internally via a suitable space group, being build on the desired point group. This also means that we do not have C_∞ and icosahedral groups. For these use proper subgroups.

4.4.6.2 Wyckoff positions

For crystals the Wyckoff positions are relative coordinates according to the standard crystallographic conventions. For molecules the positions are given in absolute Cartesian coordinates.

Alt-W to select the Wyckoff position table.

Cursor keys Move around with the help of the cursor keys.

Editing Start editing above any item by just typing.

F2 Starts editing with the old value selected. Use cursors, Del, Backspace ...

When not editing...

Ins/Ctrl-I will insert a new row below the current.

Del/Ctrl-D will delete the current row.

Ctrl+up/down will move the selected row around

Once in editing mode use

Esc to leave the editor without commiting changes!

Enter to finish editing (on Mac that is restricted)

Ctrl+A to select all

Ctrl+C to copy the value

Ctrl+V to paste previously copied values

Tab to move to the next table item

In order to enter one complicated value in more then one position use these keys!

4.4.7 Boundary Dialog

The displayed items are restricted to the boundary defined here. The boundary is either the Wigner-Seitz cell (for Fermi surfaces) or a simple box. The box is defined by giving three vectors V1, V2 and V3. The vectors are expressed in terms of either the primitive, conventional or cartesian basis vectors.

The cell's diagonal goes from (0,0,0) to V1+V2+V3. If the center-around-origin checkbox is checked, the cell's center is at (0,0,0), unless a shift vector is defined. The shift vector defines an active shift of the boundary object by the amount specified in the shift vector, which again can be given in different basis systems. The "Box" means that the shift is given in terms of the box, say (1/2,1/2,1/2) would be the box's center.

When being in a line-edit

Ctrl+A select all

Ctrl+C copy selection

Tab jump to next line-edit

Ctrl+V paste selection

Real numbers can be rationals: one can enter "3/8" instead of 0.375

4.4.8 Bond Dialog

Hotkeys:

Ctrl+1...Ctrl+4: Expand the bond tree view to the according level.

In the **Bonds/Poly** tree view

Right/Left: Expand/collapse the current item

Space: (un)check the current item, or open the "Bond-Poly dialog".

Any edit key: when on a root item open the "Bond-Poly dialog".

double click: when on root item open the "Bond-Poly dialog", on other items expand/collapse the item.

Enter: commit the data and update the scene

Ctrl-A: select allover

(Ctrl/Shift)click: change current selection (only the root items selection actually matters)

In the BondGroups table view

Cursor keys: move around

Ins/Ctrl-I will insert a new row below the current.

Del/Ctrl-D will delete the current row.

Ctrl-Up/Ctrl-Dwon: move column

Space: on checkbox: toggle, on color button: open color dialog, on Name column: start editing

Any key/F2: on name column: start editing

When clicking on an atom in the 3D view, the corresponding atom gets selected in the current BondGroup table in an open Bond Dialog. If you have multiple groups in the table check the other groups if you search for the settings of that atom. If more than one root-item is selected in the Bonds/Poly tree view and the Bond-Poly dialog is open clicking on an atom in the 3D view selects the properties of this atom unless the atom is not in the current selection. This allows to transfer the settings of one atom to all the atoms in the selection.

4.4.9 Bond-Poly Dialog

Here you define which atoms (bonds) are depicted. You also define polyhedra for the corresponding Wyckoff position.
The first thing is the "Atom/Bond visibility", which has four possible settings:

In Boundary: Show only atoms in the boundary as specified in the boundary dialog.

Add Missing: Show atoms outside the boundary in order to complete all bonds of all atoms of this type, which are inside the boundary

Delete Incomplete: Delete atoms for which some of the bonds point outside the boundary. The atoms are not deleted if they are needed to fulfill the "In Boundary" option for another atom.

Delete Loners: Delete atoms, which have only bonds, which point outside the boundary.

Note, that in the case of multiple bondgroups the bond definitions of the later bond group supercede those from earlier groups. Also note that a bond has two ends and that both ends can have different "Atom/Bond visibility".
If Apply is pressed the hole Bond Dialog gets commited and the scene updated. If Close is pressed the data are committed to the Bond Dialog but not to the application. Cancel (Escape) closes the Dialog. When the Dialog is open you also can click at atoms in the scene to load the data of another atom. This discards all previously made non-commited changes to this dialog.

4.4.10 Find atoms Dialog

4.4.11 Fermi energy and bands

Here we select the bands, which have to be plotted.

4.4.12 Brillouin zones: High symmetry points

The dialog allows to select the high symmetry points. You can set the point names, colors, font size and label depth. You can also pick points in the scene. Note, that all cells shown in the scene will provide pickable points. Beside points, you can insert jumps, which allows to backfold a path, which goes straight through more than one first BZ into the first BZ. A jump will consist usually of three point. Let's say there is a path $\Gamma X S$, which forms a straight line, where ΓX is in the first BZ and XS in the adjacent BZ. There is an equivalent line segment $X'S$ in the first BZ, with another X point, which is equivalent to the first X point. The path now reads $\Gamma X \langle \text{Jump} \rangle X'S$. In the bandplot this is translated into $\Gamma X S$. In other words the segment XX' is cut out and the band structure will look as if you go through the two BZ. This is most usefull for showing all high symmetry points in a single first BZ picture. The "break" in the "plus" button menu is not yet working in FPLO, so it is useless.

4.4.12.1 monoclinic lattices

In monoclinic lattices the choice of the a- and c-axis (monoclinic b-axis) are not unique. One can always transform via a combintion of

$$\begin{pmatrix} \vec{a} \\ \vec{c} \end{pmatrix} = \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{a} \\ \vec{c} \end{pmatrix}$$

and

$$\begin{pmatrix} \vec{a} \\ \vec{c} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ m & 1 \end{pmatrix} \begin{pmatrix} \vec{a} \\ \vec{c} \end{pmatrix}$$

with $n \in \mathbb{N}$, which yields cells with the same volume. The Wigner Seitz cell will depend on the geometry of the lattice, especially $\frac{a}{c}$ and β. This means that the WS cell of a very long stretched input cell (β close to zero) is equivalent to a WS cell of a transformed cell, which is much more isotropic. These transformations also transform the high symmetry points. That means that the names are not unique. We adopted the choice that the high symmtry points are named after the coordinates in the input cell, e.g. Y is always equivalent to $\frac{1}{2}\vec{a}$. However, these point might fall out of the first BZ, in which case we use the transformation to backfold them into the first BZ. So, they end up at points, which would be named differently, if the backfolded lattice was used in the first place.

4.4.13 Mesh Dialog

The Fermi surface mesh is created such that it adapts best to the symmetry of the lattice. To achieve that an automatic mesh subdivision is empoyed. Only a mesh subdivision parameter is needed. This is usually the subdivision along the x-aixs or first lattice direction. In order to reduce the calculation time for the band structure code, point group symmetry is used, If the checkbox for the irreducible part is checked. The subdivision is done such that the

resulting micor-cubes and tetrahedra are as isotropic as possible. For slabs that would be a waste. Therefore, one can overwrite individual subdivisions from the automatic determination. In the slab case that will be the z-direction,where we need only 1 subdivision. If the manual subdivision lineedits stay empty, the automatic value is used. Beware, that depending on symmetry restriction for possible values apply. For instance, in tetragonal lattices the x and y values must be equal.

4.4.14 Fat band editor

FPLO can export band weights in the files `+bweights`... This has limitations. FPLO can also export the complex coefficient matrix in `+coeff` when requested via a switch in the FEDIT bandstructure submenu. This file can be used to create custom tailored band weights by selecting certain molecular/atomic orbital projectors, which define the band weights. The resulting data are written to a file with a user specified file name, which then can be plotted in the usual way (xfbp `or fedit -bandplot`).

Another option is to not create the `+coeff` file and to put the file `=.bwdef...` (or whatever name you gave when saving the content of the fat band editor) into the `fedit bandplot` menu. Then fplo will create bandweights according to the definitions contained in this file and ignores the other settings in the bandplot menu.

If the DOS is required by the standard options (`bandplot` on and option `NO_DOS` not set) additional files `+bdos...` are created containing the LDOS corresponding to the patterns defined in this dialog.

To edit molecular band weights there is a helpful feature: if a structure view of the corresponding `=.in`-file is open, the atoms and absolute positions of a molecular orbital projector can be selected from the structure view via mouse click, see Band Weight Contrib Dialog.

Caveat: If you define let's say a molecular pattern consisting of two p_z-orbitals sitting at two sites, you have to define the relative phases (factor in the contrib dialog) in order to get bonding/anti-bonding orbitals. If you draw a little sketch it is clear that the equal phase pattern (1,1) must be the anti-bonding state. However, if the two radial functions belong to different main quantum numbers the radial functions can have inverted signs, since the sign of the radial part is defined by the behavior close to the nucleus and the additional nodes change the sign in the valence region. Hence, then the anti-phase (1,-1) gives the antibonding pattern!!!

Hotkeys in weight-tree-view:

F2/Enter/Space/Double-Click: open editor

DEL: delete item

Ctrl-Up/Ctrl-Down: move item

4.4.15 Band Weight Contrib Dialog

Edit the contribution of a single atom to the band weight definitions. If it is a molecular orbital only the name can be edited. Otherwise more information needs to be defined. If it is a single atom weight definition, the position is determined automatically. If it is a contrib to a molecular pattern, the absolute position has to be given. In order to facilitate this you can use the **getAtom** or **getPosition** buttons. In order for this to work a structure view of the corresponding compound has to be open. Note, that there is no cross check weither the structure belongs to the current data or not. If a valid `+coeff` exists site and orbital information is taken from this file and put into comboboxes. If the orbital information is absent, autocompletion in the orbital-lineedit is provided.

Valid orbitals can be "all", "allnlm", "3d","3d-1", "3d3/2" (relativistic) and "3d5/2/-1/2" (relativistic) if it is a single atom definition. If it is a contrib to a pattern only fully qualified orbitals ("nlm"or "$nlj\mu$") are valid. The local axis are checked for orthogonality. If they are not orthogonal it will be adjusted. If they are coplanar an error is issued.

Important: In relativistic mode pseudo non-relativistic symmetries ("nl" and "nlm") can be used. In this case the underlying colinear approximation of the full relativistic FPLO implementation requires that the resulting spin axis points along the quantization axis of the exchange field. That means that although this tool allows to select the local quantization axis for harmonics of the pseudo non relativistic orbitals, the spin projection always follows the field axis. Keep this in mind when interpreting the "spin-up" and "spin-down" LDOS/weights. Furtermore, beware that the pseudo non-relativistic projections are approximate!

The "all" orbital descriptor produces $nlj\mu$ weights and "allnlm" produces $nlm\sigma$ weights in full relativistic mode.

4.4.16 Unfold Editor

Here the unfolding information is edited. The right side is a text editor, in which the content can be freely edited. One can load and save the file.

The unfolding is explained in detail in `unfolddoc.pdf`. The information here specifies, which super cell sites are considered translationally equivalent with respect to normal cell translations. The list contains lines. Each line contains an identifier, which best is chosen to be the site number in the normal cell, which is considered a representative of the translationally eqivalent super cell sites, followed by a list of all super cell sites, which are equivalent. The list of equivalent sites can by incomplete, which leads to partial unfolding.

Additionally, there is a possibility to define the relation between the normal and super cell by defining the transformation matrix from one to the other. If the combobox "large cell in small cell units is choosen" the matrix should be integer. In the other case the matrix usually is rational. The "change on units" checkbox influences the matrix on changing the combobox setting. Either the matrix is transformed/inverted such that the new matrix and the new combo setting define the same cell relation. Or the matrix is left alone, when the combo setting is

changed. The tolerance determines if super cell atoms, which are slightly displaced from their ideal normal cell position are considered translational equivalent with respect to the translation vectors given by the matrix. The Apply button calculates the site lists according to the matrix and tolerance settings.

Note: that different elements can be considered as being translationally equivalent as long as their basis is equivalent. e.g. Fe and Ni. The automatic creation explaine in the last paragraph does not know about the basis and hence puts different elements into different unfolding definitions. If you know what you are doing you can reshuffle the unfolding definitions by hand. FPLO will abort with an error, if unfolding of different elements in the same site list is impossible.

4.4.17 Annotations

There are several kinds of objects, which can be shown inthe scene.

4.4.17.1 Axes

Axes are tripods.

4.4.17.2 Display cells

Display cells are polygonal cells. The first ("Main boundary") is special in that it defines the clipping volume for atoms and Fremi surfaces. The second and the third are the conventional and primitive cell respectively. The user can add more cells, which then have to be edited accordingly. The dialog does not allow to edit the conventional and primitive cell.

The main boundary is edited in the boundary dialog. That is so to emphasize it's special purpose.

The visual properties can be set. The visibility can be toggled (not the special hotkeys for the first three cells.)

The add (CTRL+I, Insert)and delete (CTRL+D, Delete) buttons are active when in the focus is in the cells-section of the Annotations list view.

4.4.18 Fog Dialog

Fog can be added to the sceene. For perspective view linear fog is best. The other fogs can make the scene to appear totally white. The color is probably best set to white.

The fog is determined by parameter

Exponential/Gaussian: The clearness decays with an exponential or gaussian of decay rate "**density**".

Linear: The scene is clear at "near" and becomes foggier up to "far", where the fog has the full fog color. "far" can be larger than one, which means that the fog will not reach maximum saturation at the object being furthest away. The closest object is at 0 and the furthest away one at 1;

4.4.19 Atom properties

Atoms can have colors and radii. The Tab "Themes" allows to define different themes, containing default properties for each element. The button "**Reset to app default**" resets the whole table of the current theme to some standard values. The **Accept/Close** button saves the changes and makes the current theme the default for all actions, which require atom properties, like adding Wyckoff positions to the Symetry Dialog, or opening FPLO input files like `=.in/=.sym` (which do not contain color/radius information). The button "**Save and Apply**" saves the settings and applies the current theme to the current Wyckoff positions and sites. Note, that the current (all) theme(s) is remembered persistently in the file `$HOME/.xfplo/atomprops.ini`.

In the symmetry menu the color/radii for each Wyckoff position can be Set individually. Additionally, there is the "set Default" button, which can be used to set the color/radii of all Wyckoff positions to the current theme's values. If the file "=.in" is loaded the current theme (combobox "theme") is used to define the atom properties. The properties are saved if the file =.xstr is saved. So, there are two sets of values. 1: in =.xstr 2: in the themes. The =.xstr settings supersede the theme settings on loading, unles new atoms are created.

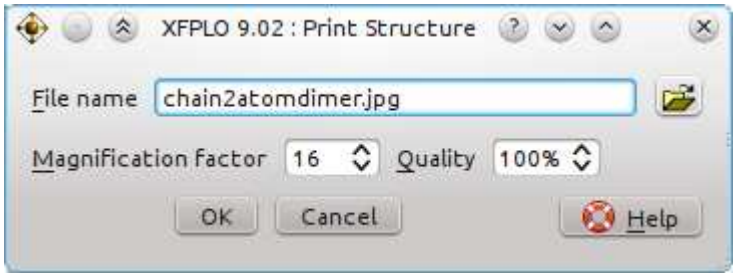
Editing: in the table there is the radius type. To edit it, either type F2 or simply the letters (cimvc). The radius field is updated and can be edited. All radii are remembered, such that one can easily switch them. Radii are given in Angstroem. Note, that the displayed atom radii are half of the actiual radii. The "Bond Dialog" contains a scale factor, which scales all atoms uniformly.

Specular color: the spot light color combines with the specular color of an objetc to create a highlighted spot.

Shine: the smaller the shine value the more diffuse the spot will be. Maximum value is 128. Imagine a metallic pollished object, in this case the spot will be very focused, while for a more diffuse surface the spot is smeared. This is why shininess produces a smaller spot for larger values.

4.4.20 Printing

Currently, printing is possible to jpg and bmp files only. The scene is printed by piecing together several enlarged shots of parts of the total scene. In this way you get a higher pixel density than depicted on screen. E.g. a magnification factor of 4 will glue together 16 pieces, resulting in a 4 times higher pixel density then on screen. Together with antialiasing this gives high quality pictures.



While printing it may happen that the intermidiate enlarged shots are displayed on screen. This is ugly but not a bug.

Printing can be interupted using the stop button close to the progress bar in the status bar.

4.4.21 Preferences

4.4.21.1 Graphics

Resolution Each display has different graphic capabilities. This might effect performence, especially while moving the scene. Several options are implemented to ease the pain.

Low resolution: Atoms and bonds can be drawn with less polygonal resolution.

Wireframe: Some graphics are faster with wireframes.

No antialiasing: Anitaliasing, when allowed, will remove some of the pixelation effects. It is however more expensive. With a jitter of 4, for example, the scene is drawn 4 times at every redraw request (moving). You can switch it of during moving. See Antialiasing for more control.

Inaccurate transparency: Transparency requires sorting of the polygons, which is slow. With inaccurate transparecny this sorting is switched off while moving.

Antialiasing Anitaliasing removes the pixalation effects by jittering the scene, which basically redraws the scene by superposing several single paints with sub-pixel offsets.

Jitter: gives the number of redraws.

For printing only: uses antialiasing only when printing the scene and not on the display. This gives faster graphics on screen. You can find a compromise using the resolution option No antialiasing.

4.4.21.2 External

For some tools external editing is possible. Give the editor and check the “run in terminal” box, if needed (e.g. for vi).

4.4.21.3 OpenGL

The OpenGL settings found here, were at least one times critical on some platform.

Vertex arrays: Vertex arrays are unlimited per OpenGL definition. Some platforms, however have limits, which on top of all this are not queryable. Hence, the only way out is to not use vertex arrays on such platforms (mostly old graphics cards or software emulators). You will find this out, if the Fermi surface is not plotet completely, when using more $\boldsymbol{k}$-points.

4.4.22 Manage executables

The fplo executables are selected by looking for filenames containing `fplo` in paths defined by environment variables (such as `$PATH`) or in selected directories or explicitly specified files. This dialog allows to compile a list of such search patterns.

The actual list of executables returned by the search algorithm in the run dialog puts the executables, matching the version of the current `=.in` file the best, at the top of the list. If there has been a previous run, the program remembers the executable used for this `=.in` and selects it. Otherwise the best match is selected.

4.4.23 Themes

Some data are more generally useful than just for a single session. Among them are atom properties and lighting. The atom properties are explaind in a separate section. The save button opens a save dialog.

If the save button is pressed, the current light settings is saved under the name given in the combobox. At the same time this theme is set to be the default light theme for each session, which does not provide previously saved light settings (loading =.in/=.sym, or starting fresh) until changed again. The Combobox allows to edit the name of the theme (only when save pressed). You can add a new theme and delete any theme (but the first). The load-theme dialog

allows to select an existing theme and load it into the current light dialog. At the same time the current theme is set as the default for each session. The themes are stored in `.xfplo/lightthemes.ini`.

4.4.24 Lighting

Lighting is a subtle issue. The program comes with a standard lighting, which works for all scenes. There is an option to define and save predefined lighting, which gets reused unless =.xstr or =.xef files are loaded (they contain their own light settings). Themes can be easily loaded via the theme load and save buttons. (see LightThemes)

For starters you can define the background color and the global ambient light. Ambient light is lighting the scene evenly. Use it with caution. Try setting it to 1 and see what happens.

There are three different sorts of lights. All lights have a diffuse color, which is diffusely reflected according to the angle between the surface of an object and the light position. Each ligth can have ambient light, which usually is better set via the global ambient light. The last is the specular light, which defines the reflection spots. The effect is defined by the specular color of the light and the specular color of the objects material.

Directional lights: Directional lights shine from a certain direction given in the position spinboxes. The light comes from an infinitly remote point and has parallel rays. So it lights all surfaces in the same angle. It is well suited for achieving a basic lighting. Displacing it away from $(x, y) = (0, 0)$ make the light spot (if specular color is not black) appear sideways. The z-coordinate is relative to the camera position to ensure working light for all scenes. Positive z means that the light sits at or behind the camera in outward direction from the screen.

Positional lights: These lights sit at a definite position in space. In the program this position is defined in units of the scene dimensions. All coordinates are measured relative to the focal point, which is in the middle of the view area and usually in the middle of the physcial scene. Positive z coordinates mean that the light sits closer to the viewer than the focal point. Negative z-coordinates put the light deeper into the scene (lighting from behind). Positional lights shine in radial rays from the position. The positional light at 000 usually sits in the center of the scene at the focus point and shines in all directions. The light diminishes the further away the object is from the light, which is controled by an attenuation factor, given by falloff $\propto \frac{1}{c_0+c_1 d+c_2 d^2}$, where d is the distance. The most important coefficient is the constant coefficient, which scales the light strength. Positional lights have another paramter w, which controls the allover distance of the light. The real position is given by $\vec{p} = \frac{1}{w}(x, y, z)$. Hence, the larger w the closer the light will be to the scene. Default is $w = 1$. You can get the same effect, by changing x, y and z by the same factor, but w is more convinient.

Spot lights: Spot lights are like positional lights, except for an added characteristic, of beging confined to a cone going out of the position into the direction specified in the dialog. The cone opening is controled by a cutoff angle, which must be in the intervall $[0, 90]$. It defines the half-angle of the cone opening. Outside the cone there will be no light. Additionally the light is dimed from the center axis of the cone towards higher angles via the exponent.

Warnings:

1. Do not use too much diffuse/specular color. Often good light settings have quite gray diffuse colors and more white-ish specular colors. The reason is that all lights together should not overexpose the scene. Ambient color is best used from the global ambient settings at the top of the dialog.
2. Too bright settings (small exponent, and cutoffs (opening angles)) give ripple effects on approximated surfaces. So use wider spot lights, with larger exponents and correct with constant attenuation.
3. Dont underestimate the power of the constant attenuation in connection with positional/spot lights. Going below the default value 1 can give results in combination with the ripple avoidance, discussed above.

Strategy: In order to achieve good lighting the following tips are helpfull.

1. Use the default.
2. There are light helpers, which can be switched on, which are not perfect, but help orient.
3. Use perspective viewing for better orientation, when setting up lights.
4. Switch off all lights but one, in order to get somewhere. Start from position 000 and reasonable direction (spot lights) (the default direction $00-1$ points into the screen.)
5. Directional lights with too negative z are most likely not much visible.
6. Try small changes of all parameters to understand the effects and the light helpers.
7. Note, that the bright spots get reflected by the surface in an angle and hence appear at some angle between the light direction and the eye. Don't get confused by the direction the light helper points to being different from the spot position on the object.

4.4.25 Molecular/individual band weights

The default method to created an orbital projected bandstructure (fatbands/band weights) is to switch on band weights in FEDIT. The more flexible way is to use the band weight editor mode of XFPLO. On a fresh start use XFPLO `-bw` or with an existing file XFPLO `=.bwdef` to edit the desired band weights. There are help screens in the program.

One can use this interface to request single weights for selected atoms, which would help to reduce the file size of the usually big file `+bweights`.

One can also create molecular orbital projectors, which are linear combinations of atomic orbitals with certain coefficients, which determine the bonding character and symmetry of the molecular orbital.

4.4.26 Structure viewer

Structures can be loaded via

```
XFPLO =.in
XFPLO =.sym
XFPLO =.xstr
```

Fresh start in structure mode

```
XFPLO -str
```

Structures can be manipulated in the symmetry dialog. From there one can export `=.sym`, which will also give the option to update `=.in`. The whole picture (not just the structure) can be saved into `=.xstr`.

4.4.27 Fermi surface viewer

A fresh start in fermi surface mode is

```
XFPLO -fs
```

One can save all settings (but not the +band data ... to big!) in `=.xef`, which can be loaded via

```
XFPLO =.xef
```

One can also define the path through the Brillouin zone for band structure plots in the Fermi surface mode. To achieve this first you need at least a `=.in` (use FEDIT or the symmetry dialog in the structure viewer to create this.) Then open

```
XFPLO -fs
```

Now, switch off the Fermi surface display via the Fermi-surface button and switch on the high symmetry points (Fig. 4.1). Open a dialog: Menu plot → high-symmetry-points. Switch on user-defined. Load default if you want. Select a point in the list. Click on the pick button,

select current, click on a green point in the Brillouin zone, hit enter or press accept. You must have changed a point. Use pick-button → to create a new point. Use F2 in the list to edit or just start typing. Export to or import from `=.in`, if needed.

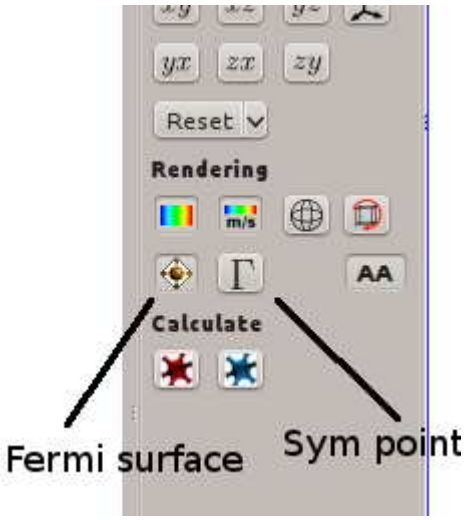


Figure 4.1: Fermi-surface and sym-points buttons.

4.4.28 *Density mapper*

This is not well tested and was implemented on special request of some user.

An existing calculation (call it A) with a `=.dens` file can be a good starting point for a modified structure (call it B). However, since the structure of B determins the dens-file of B the dens-file of A cannot be used in B. You can use XFPLO to define the file `=.densmap`.

We assume that calculation A exists and that the structure of calculation B is already set up (`=.in` and `=.sym` exist).

1. Go into the directory of the the new calculation B.
2. Call "XFPLO `=.in`" to display the structure.
3. Open tools→density-maper.
4. Click button "Open old structure" and select the `=.in` of directory A. This will open a new structure view, this time of A.
5. In the mapper table click on an atom of new structure B. The atom will be highlighted. Now click on the atom fo structure A, whose density shall be compied onto the selected B-atom in the dens-file to be created.

6. You can use the "Copy this atom" button to copy this definition to all B-atoms with the same element. Proceed with all B-atoms, which have an equivalent in the A-structure. You can leave some B-atoms un-mapped, in which case a default starting density for these missing atoms will be produced by FPLO. You can also open other old structures and map atoms from there.
7. Save the file in the directory of calculation B (name `=.densmap`).
8. Quit XFPLO.
9. Run FPLO in the directory of B until it stops after creating a new `=.dens`.
10. Zip or rename or delete `=.densmap`.
11. You can now start the B calculation with the freshly mapped dens-file.

Note, that only the spherical part of the local site densities are copied during mapping. Hence, the created mapped density is not nessecarily a good starting point.

4.5 Plotting program: XFBP

Use this to plot band structures/DOS and stuff like this. There are help screens (main window and Script editor (Transform Dialog), which explain basic operations:

4.5.1 Scripting

Contents

The Transform Dialog is actually a script editor, where XFBP commands from the file description language and some more commands can be used. The commands can be saved to file or loaded from file (extension `.cmd`). To get familiar with the commands you can use the insert functionality of the editor. We also recommend to look at the `.xfp` files: save the current plot (`.xfp`) (not the current script) and look at the `.xfp` file.

We try to describe the scripting language in the following, including examples. To copy these examples into the script editor, select and copy them via Ctrl-C and paste them (Ctrl-V) into the script editor.

The language is made of statements. Each statement is a single line.

In the following optional parts of a statement are enclosed in square brackets []. A construct like (a|b|c|...) means that at this position in the command either a or b or c ... can be used. If parentheses "(" and ")" appear without a "|" between them they are literals, which means you have to type them (example: `sin(x)`). They will be set in bold face in the command description. Similarly square brackets "[" and "]" can appear in vector element constructs (example: `s1.x[6]=12.5`).

In the following key words are denoted by **keyword**. They are not case sensitive. Values are denoted by *value*. There are different values:

- sub commands like *linestylecommands*
- expressions like *exp*, which can be numbers, variables, parameters and some other constructs. They evaluate to scalars or vectors
- *string*s, which are used for file names and text. They must be enclosed within double quotes, example:

```
read xny "+dos.total"
read bandweight "+bweights"
```

- *parameter*s are values defined on the command line via option `-a` and available in the script editor via *$parametername*. Example: Create a command file `t.cmd` with the content

```
kill all
read xny $pp
with g1.gr1.s1
line width $w
legend $leg
title "Density of states"
```

From the command line call

```
xfbp t.cmd -a pp:"+dos.sort001" -a leg:"Fe" -a w:2
```

Of course, you have to make sure that there is actually a file `+dos.sort001` for this to work.

Important: when graphs, groups, sets or weights are referenced in the script, they will be made current, which means that they will be remembered until the next such explicit reference makes another graph... the current object. This allows to skip these specifiers in many contexts.

File loading
Print commands
Paper commands
Graph/Group/Set/Weight-descriptors
With command
World commands
View commands
Legend box commands
Graph commands
Shape commands
Text box commands
Tick mark commands
Group commands
Set commands
Set attribute commands
Weight commands
Weight settings
Weight label definitions
Line style commands
Fill style commands
Font style commands
Symbol style commands
Kill commands
Copy/Move commands
Hook commands
Cursor reference
Assignements/Definitions
Expressions
Derivatives

Comments

- # blah blah

  Full line comment. There are no inline comments!

File loading:

- **read** *filetype* (*string*|*parameter*) [(**into** (*graphdesc*|*groupdesc*)|**into new graph**)]

  Read file with name *string* or with its name provided in a *parameter* into current graph (or a specific graph/group) enforcing data type *filetype*. (*string*|*parameter*) is something like "myfile.dat". Band structure data cannot be read into specified groups, since they are organized into groups by the programm.

  Examples:

```
# killall and initialize graph 1
killall
# read band strcture into current graph (graph 1)
read band "+band"
# read two files into graph 2
read xny "+dos.total" into g2
read xny "+dos.sort001" into g2.gr5
```

Top

Print commands Currently there is only one option to print: eps-files. The printed format is not yet fully eps standard, but works, especially embedded in LaTeX.

- **print to** (*string*|*parameter*)

  export plot to file named *string* in eps format. Example:

```
print to "bands.eps"
```

- **print filename** (*string*|*parameter*)

  set the the file name for print commands to *string* or *parameter*.

- **print to file**

  export plot in eps format to file. (The filename must have been defined before.)

Top

Paper commands:

- **paper size** *int* , *int*

  set paper size from width and height (integer). Example:

```
paper size 400,200
```

- **paper size** *papersizes*

  set paper size from paper size names, e.g. a1...a10, letter, lettersmall, legal, statement, tabloid, ledger, folio, quarto. Example:

```
paper size a4
paper orientation portrait
```

- **paper orientation** (**portrait**|**landscape**)

  set orientation. Only for predefined paper sizes.

Top

Graph/Group/Set/Weight descriptors: Data belong to graphs and are organized in groups and sets. graph/group/set/weights have descriptors, which are explained here. Note, that there is the concept of a current graph/group/set/weight. The current object is memorized after a descriptor appeared in the script until another object becomes current by the appearance of another descriptor . Hence we can write

```
g1.gr2.s1 line color 0xff
line width 2.5
line symbol ...
```

to change the settins of set 1 in group2 in graph 1.

- **g***int*

  graph *int*: g4 is graph 4.

Table of Contents | Index

- **gr***int*

  group *int* in the current graph (if valid)

- **s***int*

  set *int* in the current graph and current group

- **g***int1* . **gr***int2*

  group *int2* in graph *int1*

- [[**g***int1* .]**gr***int2* .]**s***int3*

  set *int3* in current group or group *int2* in current graph or graph *int1*

- [(*groupdesc*|*setdesc*) .](**x**|**y**)

  vector of x/y values in groupdesc/setdesc or current group/set. Example: `g1.gr1.x=x+1`, will increase all x values of all sets in `g1.gr1` by one. The full command would be `g1.gr1.x=g1.gr1.x+1`, but this is not necessary. And `g1.gr1.s2.y=y*2`, will multiply all y values of set `g1.gr1.s2` by two.

Note, that each descriptor resets the higher level to invalid, such that following works:

```
g1.gr1.s3 line color 0x00
# now  current set is s3, current group is gr1
line width 2.4
# it is still set 3
g1.gr1.x=mesh(...)
# after g1.gr1 the current set is invalid and hence all x of group1 get assigne  a mesh
line width 2.4
# now we just set the line width for all sets in group 1
```

Each set can have a number of weights associated with them. If all sets of a group refer to the same set of weights (same set of weight labels.) the group can be used to manipulate the weight appearance. (bandweight plots)

- [(*groupdesc*|*setdesc*) .]**w***int*

  weight number *int* in groupdesc/setdesc or current group or set (whichever was defined/refered-to previously). Example:

  ```
  g1.gr1.w8 off
  # turns off weight 8 in group 1.
  ```

- [(*groupdesc*|*setdesc*) .]**w***string*

  weight with name *string* in specified or current group/set. The names must match the existing weight labels exactly. Example:

  ```
  w"Cu(001)3s+0" off.
  ```

Top

Table of Contents | Index

With command The current set/graph/group/weight descriptors can be set without reference to a particular commmand.

- **with** (*graphdesc*|*groupdesc*|*setdesc*|*weightdesc*)

  set the current graph/group/set/weight. Example:

```
with g1.gr1.s3
line color 0xff0000
symbol style circle
# now weigths with implicit with (remembered from the last weightdescriptor)
w1 off
w"Cu(001)3d+0" on
# set this weights color
color 0xff0000
# and the skippage
skip 5
```

Top

World commands World commands refer to a certain graph, which either is specified explicitly or was set as current graph before.

- [*graphdesc*] **world** (**xmin**|**xmax**|**ymin**|**ymax**) *exp*

  set world $x_{\min}$, $x_{\max}$, $y_{\min}$ or $y_{\max}$ for graph. Example:

```
world xmin -1
```

- [*graphdesc*] **world** (**x**|**y**) *exp* , *exp*

  set the x or y interval of the world coordinates. Example:

```
world x -1,10
```

- **autoscale** [(**x**|**y**)]

  autoscale all, x-only or y-only. Example:

```
autoscale x
```

- **autoscale offset** *exp* , *exp*

  define the autoscale offset for both directions. This is the space in percent of the total world interval, which will be added on both sides of the intervall. If non-zero, an autoscale will leave the specified percentage of space at either side of the curves plotted.

The current settings can be loaded from the insert menu in the script editor.

Top

View commands The view is the area spanned by the frame of the graph. It is defined in relative coordinates with respect to the paper. A view of width 1 and x-origin 0 would produce a graph frame spanning the whole breadth of the paper. In the following graphdesc can be left out if previously a graph was referenced (**with g***int*, or other references).

- [*graphdesc*] **view** *exp* , *exp* , *exp* , *exp*

  set x-origin, width, y-origin and height of the view. x,y=(0,0) is the upper left corner and x,y=(1,1) the lower right corner. *exp* must evaluate to a scalar.

- [*graphdesc*] **view frame** (**on**|**off**)

  switch frame of view on or off

- [*graphdesc*] **view frame rim** *linestylecommands*

  use *linestylecommands* for the rim of the view frame

  [*graphdesc*] **view frame fill** *fillstylecommands*

  use *fillstylecommands* for the view frame filling

The current settings can be loaded from the insert menu in the script editor.

Top

Legend box commands The legend box belongs to a particular graph. The following commands refer to the current graph. They can also be prefixed with a *graphdesc* or follow after **with g***int* to set the current graph (as indicated in *graphcommands*).

- **legend** (**on**|**off**)

  switch legend box on or off

- **legend font** *fontstylecommands*

  use *fontstylecommands* for text in legendbox

- **legend line spacing** *exp*

  set spacing between legend box entries (lines). *exp* must evaluate to a scalar.

- **legend symbol marker spacing** *exp*

  set spacing between symbol and text

- **legend symbol marker width** *exp*

  set width of symbol before the text

- **legend frame** (**on**|**off**)

  switch frame on or off

- **legend frame rim** *linestylecommands*

  use *linestylecommands* for the rim

- **legend frame border spacing** *exp*

  set spacing between rim (border) and content (symbols, text)

- **legend frame fill** *fillstylecommands*

  use *fillstylecommands* for the frame

- **legend position** *exp* , *exp*

  set legend position relative to the view frame (y=0 is on the top). The position refers to the point of the legend box given as origin (see below)

- **legend origin** *exp* , *exp*

  define the point of the legend box, which is considered its origin. (the **position** command places this point.)

The current settings can be loaded from the insert menu in the script editor.

Top

Graph commands Graph commands will manipulate graphs. A graph is a single view-frame/plot with axes, which can contain several groups/sets. Graphs have (irregular) tick mark settings and can contain shapes (lines/ellipses). Graphs can be scaled.

- *graphdesc* (**on**|**off**|**toggle**)

  switch graph on or off or toggle on/off (for hook commands). Example:

  ```
  g2 on
  ```

- **new graph**

  create a new graph. More control is achieved via switching a particular graph on or off.

- [*graphdesc*] *textboxcommands*

  use *textboxcommands* for the current/specified graph. Example:

  ```
  g1 subtitle on
  subtitle "some subtitle"
  subtitle font color 0xff
  ```

- [*graphdesc*] *legendboxcommands*

  use *legendboxcommands* for the current/specified graph.

- [*graphdesc*] *tickmarkcommands*

  use *tickmarkcommands* for the current/specified graph.

- [*graphdesc*] *irregulartickmarkcommands*

  use *irregulartickmarkcommands* for the current/specified graph.

- [*graphdesc*] **graph line width scale** *exp*

  set an overall line width scale for the current/specified graph. This affects all line widths.

- [*graphdesc*] **graph point size scale** *exp*

  set an overall point size scale for the current/specified graph. This affects font and symbol sizes and line widths.

- [*graphdesc*] *shapecommands*

  manipulate shapes (arrows/lines/ellipses), see *shapecommands*. Example:

```
g1 line1 on
line1 line width 2
```

Top

Shape commands In the moment line shapes (lines/arrows) and ellipses (circles) can be added to a graph. First let us define shape descriptors:

- **ellipse**[ ]*int*

  reference to ellipse number *int*. The space between the keyword and the number is optional. Example:

```
ellipse3
```

- **line**[ ]*int*

  reference to line shape number *int*. This can be an arrow not just a line. The space between the keyword and the number is optional. Example:

```
line 2
```

Now, we give the details for each shape type. Some commands are specific for the particular shape kind others are common.

- *shapedesc* (**on**|**off**)

  switch on (create it of not yet existing) or off a shape. Example:

```
line1 on
```

- *shapedesc* **name** (*string*|*parameter*)

  give the shape a name. This is usefull in the GUI, where all shapes appear in a list.Example:

```
line1 name "my-fancy-line-name"
```

- *shapedesc* **line** *linestylecommands*

  use *linestylecommands* for the shape's rim/line.Example:

  ```
  line1 line color 0xff
  line1 line style dot
  ```

- *shapedesc* **coordinatesystem** (**view**|**world**)

  use this coordinate system's units for the shape's size/positional settings. World coordinates refer to the physical coordinates of the underlying plot. Hence, changing the zoom of the plot will move the shape with it. View coordinates refer to the graph's view/frame. These are rleative coordinates, where x,y=(0,0) is the upper left corner and x,y=(1,1) is the lower right corner of the view.

- *ellipsedesc* **fill** *fillstylecommands*

  use *fillstylecommands* for the ellipse's interior.Example:

  ```
  ellipse1 fill color 0xff
  ellipse1 fill on
  ```

- *ellipsedesc* **angle** *exp*

  set the angle about which the ellipse is rotated.

- *ellipsedesc* **center** *exp* , *exp*

  set the center of the ellipse. This refers either to world or view coordinates.

- *ellipsedesc* **radii** *exp* , *exp*

  set the two radii of the ellipse. This refers either to world or view coordinates.

- *ellipsedesc* **radius** *exp*

  set both radii of the ellipse to the same value (making it a circle). This refers either to world or view coordinates.

- *linedesc* **arrow fill** *fillstylecommands*

  use *fillstylecommands* for the line. Note, that only the fill color command does work here, i.e. closed arrow heads allways have "`fill on`". If an empty head is required, use `fill color 0xffffff` (white).

- *linedesc* (**arrow**|**cap**) **position** (**none**|**start**|**end**|**both**)

  set the position of the arrow head or cap on the line shape. A cap is a little runding off at the end of the line. better visible if the line width is larger.

- *linedesc* **arrow style** (**open**|**closed**)

  open arrows are made of lines, closed ones are haveing a filling.

- *linedesc* **arrow** (**size**|**sharpness**) *exp*

  set the size or sharpness of the arrow head.

- *linedesc* (**start**|**end**) *exp* , *exp*

  set the start and the end of the line shape. (refers to the specified coordinate system.)

The current settings of existing shapes can be loaded from the insert menu in the script editor.

Top

Text box commands Text boxes are used for axis labels but can also be placed freely in the graphs. They belong to a particular graph. The special text boxes like axes- and tick labels can impose restrictions on each others placement, which are used to keep them at reasonable distance from each other. Text boxes have an origin, with respect to which the position is defined.

Textboxes are identified by a descriptor:

- **textbox**[ ]*int*

  the general text box number *int*. The space is optional. Example:

```
textbox1
#or
textbox 1
```

- (**title**|**subtitle**|**xaxislabel**|**yaxislabel**|**oppositexaxislabel**|**oppositeyaxislabel**)

  one of the special textboxes

Now, the commands:

- *textboxdesc* (**on**|**off**)

  switch textbox on or off

- *textboxdesc* (*string*|*parameter*)

  set the content of textbox to *string* or *parameter*. The *string* can contain formating. Example:

```
title "N$_0$.$x{-0.7}$^$iFe$n$."
```

  will produce N_0^{Fe}

- *textboxdesc* **font** *fontstylecommands*

  use *fontstylecommands* for textbox. Example:

```
title font color 0xff00ff
```

- *textboxdesc* **frame** (**on**|**off**)

  switch this textbox's frame on or off.

- *textboxdesc* **frame rim** *linestylecommands*

  use *linestylecommands* for the frame rim of textbox

- *textboxdesc* **frame fill** *fillstylecommands*

  use *fillstylecommands* for the filling of textbox's frame

- *textboxdesc* **frame border spacing** *exp*

  define the space between the frame rim and the text.

- *textboxdesc* **coordinatesystem** (**view**|**world**)

  Set the coordinate system for the text position.

- *textboxdesc* **position** *exp* , *exp*

  set the position of the textbox. This referes to the specified coordinate system. See origin below.

- *textboxdesc* **origin** *exp* , *exp*

  The textbox has an origin, which is positioned accoding to the **position** command.

- *textboxdesc* **angle** *exp*

  set the rotation angle of the textbox

- *textboxdesc* **restriction** (**+x**|**-x**|**+y**|**-y**) *exp*

  restrict the movement of this textbox to the positive/negative x/y-axis and shift it in this direction by the amount *exp*. Example:

  ```
  title  restriction -Y 0.03
  ```

- *textboxdesc* **restriction none**

  define that there are no additional placement restrictions for this textbox

- *textboxdesc* **restriction additional** [*int* [*int*[*int*. . .]]]

  define a list of *int* values, encoding various additional restrictions refering to the special text boxes. The list can be empty.

x-tick labels	*int*	**y-tick labels**	*int*	**sub-title/axis-labels**	*int*
opposite x tick label height	1	opposite y tick label width	5	subtitle offset	9
opposite x tick label offset	2	opposite y tick label offset	6	subtitle height	10
normal x tick label height	3	normal y tick label width	7	opposite x-axis label offset	11
normal x tick label offset	4	normal y tick label offset	8	opposite x-axis label height	12

The current settings of existing textboxes can be loaded from the insert menu in the script editor.

Top

Tick mark commands Ticks come in two varieties: regular ticks, which are graph specific and irregular ticks which can belong to a graph or a group.

Regular tick commands:

- (**x**|**y**) **auto tic**[**s**] (**on**|**off**)

  auto ticks on means that the tick spacing is determined automatically, off means that the user has to set the major tick spacing and the minor tick subdivisions. Example:

  ```
  x auto tics off
  ```

- (**x**|**y**) **axis scaling** (**log**[**arithmic**]|**lin**[**ear**])

  set the axis scaling to linear or logarithmic. Also read Logarithmic plots. Example:

  ```
  x axis scaling log
  ```

- (**x**|**y**) **tic**[**s**] **side** (**none**|**normal**|**opposite**|**both**)

  on which side to place the ticks. Example:

  ```
  x tic side opposite
  ```

- (**x**|**y**) (**major**|**minor**) **tic**[**s**] (**on**|**off**)

  switch the drawing of x/y major/minor ticks on or off. Example:

  ```
  x minor tics off
  ```

- (**x**|**y**) **major tic**[**s**] **spacing** *exp*

  the distance of major ticks in world units. For logarithmic axes the spacing between two major ticks is determined by multiplication with this number instead. Example:

  ```
  x major tic spacing 0.1
  ```

- **y** (**major**|**minor**) **separate tic**[**s**] **length** (**on**|**off**)

  if switched on, the length of the y axis ticks is set separatly. Otherwise it is the same physcial length as for the x-axis ticks. Example:

  ```
  y major separate tic length on
  x major tic length 0.03
  y major tic length 0.05
  ```

- (**x**|**y**) (**major**|**minor**) **tic**[**s**] **length** *exp*

  set the length of the ticks in view units.

- (**x**|**y**) **minor tic**[**s**] **subdiv** *int*

  set the number of subdivisions of a major tick interval to define the position of the minor ticks.Example:

  ```
  x minor tic subdiv 2
  ```

- (**x**|**y**) (**major**|**minor**) **tic**[**s**] **line** *linestylecommands*

  use *linestylecommands* for the ticks. Example:

  ```
  x major tic line color 0xff00
  ```

- (**x**|**y**) **tic**[**s**] **label**[**s**] **decimals** *int*

  set the number of decimals to be printed in the tick labels. A negative number signals to use the automatic default. Example:

  ```
  x tic label decimals 2
  ```

- (**x**|**y**) **tic**[**s**] **label**[**s**] **offset** *exp*

  the label offset from the axis in view units. Example:

  ```
  x tic label offset 0.1
  ```

- (**x**|**y**) **tic**[**s**] **label**[**s**] **side** (**none**|**normal**|**opposite**|**both**)

  where to draw the tick labels. Example:

  ```
  x tic label side both
  ```

- (**x**|**y**) **tic**[**s**] **label**[**s**] **font** *fontstylecommands*

  use *fontstylecommands* for label's text. Example:

  ```
  x tic label font color 0xffff
  ```

Top

Irregular tick commands:

Irregular ticks can be owned by graphs and groups. Group owned irregular ticks will be visible if the group is visible. These commands must follow after a particular graph/group was specified by a previous command (*graphcommands*, *groupcommands*) or via a **with**-command. The group irregular tick mark commands always need a *groupdesc* prefix.

- **irregular tic**[**s**] (**on**|**off**)

  switch on irregular tics. Example:

  ```
  gr1 on
  g1.gr1 irregular tics on
  ```

The irregular tics get descriptors formed in the following way:

- **itic**[ ]*int*

  an irregular tick descriptor. The space is optional. Example:

  ```
  itic1
  ```

Now the commands

- *iticdesc* **type** (**x**|**y**) (**major**|**minor**)

  define if itic is an x or a y tick and if it is major or minor. Major tics can have labels. Example:

  ```
  itic1 type y major
  ```

- *iticdesc* **length** *exp*

  define the tick length of itic. This is in view units. a length of 1 creates a line spanning the whole view area. Example:

  ```
  # span the full
  itic1 length 1
  ```

- *iticdesc* **position** *exp*

  the tick position in world units. Example:

  ```
  itic1 position 0
  ```

- *iticdesc* **label**[**s**] (*string*|*parameter*)

  the itic's label. Only for major labels. Example:

  ```
  itic1 label "E$_F$."
  ```

- *iticdesc* **tic**[**s**] **side** (**none**|**normal**|**opposite**|**both**)

  at which side to draw the itic

- *iticdesc* **label**[**s**] **side** (**none**|**normal**|**opposite**|**both**)

  at which side to draw the itic label.

  ```
  itic1 tic side normal
  itic1 label side opposite
  ```

- *iticdesc* **line** *linestylecommands*

  use *linestylecommands* for the itic. Example:

  ```
  itic1 line style dot
  ```

Example: the Fermi level can be done like this:

```
gr1 on
g1.gr1 irregular tics on
itic1 type y major
itic1 length 1
itic1 position 0
itic1 label "$~e$_F$."
itic1 label side opposite
itic1 line style dot
```

Top

Group commands Group commands manipulate groups. They act on the current group, which can be set either explicitly or implicitly as explained in so many other places.

- [*groupdesc*] **use attributes** (**on**|**off**)

  if on, the group attributes will be used for each set of the current or specified group. When switched off, each set's individual properties will be used.

- [*groupdesc*] *setattribcommands*

  use *setattribcommands* for the current or specified group

- [*groupdesc*] *weightsettings*

  use *weightsettings* for the current or specified group

- *groupdesc irregulartickmarkcommands*

  define *irregulartickmarkcommands* for this group.

Top

Set commands Set commands are commands, which modify a single set (sometimes all sets of a group). The set can be made current via the **with** command or via at least one explicit set descriptor.

Example:

```
# we assume that the current graph and group are already set
s1 on
# alternatively
with s1
# now the properties
line color 0xff0000
symbol style circle
convolute(s1,0.5)
```

- [*setdesc*] *setattribcommands*

  use *setattribcommands* for this set

- [*setdesc*] *weightsettings*

  use *weightsettings* for this set

- [(*setdesc*|*groupdesc*) .](**x**|**y**) = *exp*

  assign *exp* to the x/y-vector of this group/set. *exp* can be a vector or scalar expression. Example:

```
# this will create a parabola by setting y[i]=x[i]^2 for each point i in t  set
g1.gr1.s1.y=x^2
# or
with g1.gr1.s1
y=x^2
```

- **convolute** (*setdesc* , *exp*)

  convolute the set with a Gaussian of half width *exp*.

- *setdesc* **length** *exp*

  set the length of the x and y vectors. (The set must by of xy-type.)

- **bspline** (*setdesc* , *int1* , *int2*)

  construct the *int2*-th derivative of the B-spline interpolation of order *int1* of the set and apply it to the set. B-splines are spline interpolations of arbitrary order. Note, that there must be a minimum number of data points to construct it. Zeroth order means histogram. First order means linear interpolation second order quadratic spline-interpolation and so on. Note, that *int2*=0 means zeroth derivative and that nothing changes, since the bspline is an exact interpolation at the data points and a zeroth derivative is an identity. To calculate derivatives you chose a spline order, which does not lead to too many wiggles but is high enough to smoothly represent the data. Example:

```
killall
with g1.gr1
s1 on
N=100
m=mesh(0,10,N)
s1 length N
x=m
y=sin(x)
line color 0xff
copy s1 to s2
```

```
s2.y=cos(x)
line color 0xff0000
#first derivative
bspline(s1,2,1)
copy s1 to s3
#difference
s3.y=y-s2.y
line color 0xff00
legend "error"
autoscale
```

- **bspline** (*setdesc* , *int1* , *int2* , *identifier*)

  construct the *int2*-th derivative of the B-spline of order *int1* and interpolate it onto the x-vector pointed to by *identifier*, which must be a vector. The result is saved in set.y. Example:

```
kill all
g1.gr1.s1 on
data xy
0 0
1 1
2 4
end data
legend "raw data"
line color 0xf00
symbol style circle
symbol fill on
line style none
N=100
m=mesh(-1,3,N)
copy s1 to s2
bspline(s2,1,0,m)
s2 legend "spline order 1"
line color 0xff0000
copy s1 to s3
bspline(s3,2,0,m)
s3 legend "spline order 2"
line color 0xff00
autoscale
```

- **integrate** (*setdesc*)

  running sum integral of set. This produces the curve $I(x) = \int_{x_0}^{x} y(u)\,du$ for each point x_i in the set (upper boundary indefinite integral). If you need the value of the integral in

a script do something like:

```
integrate(s1)
sum=y[length-1]
```

- **data** *datatype*
  datalines
  end data

  define a block of data of *datatype*

datatype	**explanantion**
xy	standard y(x)
grid/xynz	z(x,y)
xynw	y(x) and w(x,y)

  The *datalines* depend on the *datatype*. They may contain *identifier* and things like **world.xmin**. Example. Also look at saved `.xfp` files.

- [*setdesc*] **weightlabel**[**s**] **reference**[ ]*int*

  set the weight label reference of the current set to *int*. This refers to a particular *weightlabeldefinitions*, defined elswhere. Sets with the same reference are treated as having the same set of weights. This allows to manipulate the weights settings of all these sets together (especially when grouped). The number of weights and dimensions of x/y of all these sets must be the same!

Top

Set-attribute-commands Set attributes can be applied to groups or sets. The last group/set descriptor or **with** command will decide which object's attributes are defined. You can also prefix each command or at least the first of a series of such commands with an explicit group/set descriptor.

- (**on**|**off**)

  switch the current group/set on or off.

- **toggle**

  toggle the current group/set. (For hook commands)

- **line** *linestylecommands*

  use *linestylecommands* for the current group/set.

- **symbol** *symbolstylecommands*

  use *symbolstylecommands* for the current group/set.

- **legend** (*string*|*parameter*)

  set the legend string for the current group/set.

- **showin legend** (**on**|**off**)

  decide if this group/set is shown in the legend box.

- **interpolationdepth** *int*

  set the interpolation depth for grid/density plots. (**Experimental**)

- **max cutoff** *exp*

  set the cutoff for density grid/density plots. (**Experimental**)

- **use secondderivative** (**on**|**off**)

  decide the use of second derivatives for grid/density plots.(**Experimental**)

- **setcomment** *string*

  set the "set-comment" of the current group/set. Set-comments denote, where the data came from. This is mostly used in the *.xfp files, which get saved and loaded by XFBP.

Top

Weight commands Weight commands determine the appearance of individual weights of groups or sets. The internal organization of weights into groups is a bit confusing. As long as your weights came from an FPLO bandweights file it should work alright. Weight descriptors can be made current via a **with** command or by using a weightdesc (at least for the first command which belongs to a particular weight) Examples:

```
w11 on
plotorder 1
name "yx"
# or
with w11
on
plotorder 1
name "yx"
```

- [*weightdesc*] (**on**|**off**)

  switch the weight on or off. Note, that if the current weightdesc is invalid but some other descriptor is valid, this command switches the currently valid graph/group/set. Example:

  ```
  with g1
  on
  # we switched graph 1
  ```

```
with gr1.w2
# implicit g1 explicit gr1 and w2
on
# now, we switched weight 2
```

- [*weightdesc*] **plotorder** *int*

  the weight with the higher plotorder gets plotted later.

- [*weightdesc*] **name** (*string*|*parameter*)

  rename the weight. This affects the legend entry. Note, that a name change also means that the weightdesc `w''weigthname''` changes as well. Furthermore the weight label/name is unique in that it is a property of the weight itself such that if one changes the name of a weight in a group it also changes the name of this weight in all sets, which are assoziated to this weight. i.e. all sets of all groups, which are belonging together, like spin up and down groups of a +bweights plot.

- [*weightdesc*] **color** *hexconstant*

  set the weights color to *hexconstant*

- [*weightdesc*] **skip** *exp*

  skip as many symbols between plotted symbols (unless weight style is connected). skip 0 means: plot all symbols.

- [*weightdesc*] **symbol fill** (**on**|**off**)

  fill symbols or not. (only if weight style is individual).

- [*weightdesc*] **symbol line width** *exp*

  for open symbols the symbol line width matters. (only if weight style is individual).

- [*weightdesc*] **symbol style** *symboltype*

  set the *symboltype*. (only if weight style is individual).

Top

Weight settings Weight settings can be applied to groups or sets. The last group/set descriptor or **with** command will decide which object's settings are defined. Weight settings affect the appearance of all visible weights of a group or set.

- **weight max** *exp*

  set the symbol size, which represents a weight value of 1. *exp* is in a scale like font sizes.

- **weight min** *exp*

  set the weight symbol size, under which no symbols will be plotted. This depends on **max** since it scales everything up.

- **weight factor** *exp*

  scale all weights by this factor, before applying **max** and **min**. (This is somehow superfluous).

- **weight style** (**dots**|**connected**|**individual**)

  set the style of the weights.

dots	individual filled circles for each data point
connected	connect the circles by linear intepolation
individual	each weight can have its own symbol

Top

Weightlabel defintions This is special stuff to define tables of weightlabels especially in saved files.

- **weightlabel definition**[**s**]
  weightlabels[ ]*int1*
  stringlist
  end weightlabel[**s**]
  ...
  weightlabels[ ]*intn*
  stringlist
  end weightlabel[**s**]

  define tables of weightlabels with references *int1* through *intn*. See weightlabel references. *stringlist* is a list of weight labels (strings), each on a separate line.

Top

Line style commands Line style commands are used in several other commands.

- **color** *hexconstant*

  define the color using a hex number, which encodes the RGB (red-green-blue) color components. The constant contains three bytes (leading zeros need not be specified) the left-most is red, then green and right-most is blue:

hex	**color**	**red**	**green**	**blue**
`0x000000=0x0`	black	0	0	0
`0xff0000`	red	255 (ff)	0	0
`0x00ff00=0xff00`	green	0	255 (ff)	0
`0x0000ff=0xff`	blue	0	0	255 (ff)
`0x800000`	darkred	128(80)	0	0
`0xffffff`	white	255(ff)	255(ff)	255(ff)

- **width** *exp*

  set line with

- **style** (*linestyle*|*int*)

  set line style via name or number

linestyle	***int***	***linestyle***	***int***	***linestyle***	***int***
none	0			long dash	7
solid	1	dash dot	4	long dash dot	8
dash	2	dash dot dot	5	long dash dot dot	9
dot	3	dot dash dash	6	dot long dash long dash	10

  Example:

```
s1 line style dash
# or
s1 line style 2
```

Top

Fill style commands These commands appear in various other commans and usually cannot stand by themselfs (they are preceded by ... **fill**).

- **color** *hexconstant*

  define fill color using a *hexconstant*

- (**on**|**off**)

  switch filling on or off

Top

Font style commands These commands are used in several commands after **font**.

- **size** *exp*

  set the font size to *exp*, which must evaluate to a number.

- **subscriptscale** *exp*

  set the scaling down ratio of subscript

- **color** *hexconstant*

  define font color using a *hexconstant*

Top

Symbol style commands These commands are used in other commands and preceded by **symbol**.

- **style** (*symboltype*|*int*)

  set symbol style by name or number

symboltype	***int***	***symboltype***	***int***	***symboltype***	***int***
none	0	triangleup	4	plus/+	8
circle	1	triangleleft	5	cross/x	9
square	2	triangledown	6	star/*	10
diamond	3	triangleright	7		

  Example:

```
s1 symbol style square
# or
s1 symbol style 2
```

- **size** *exp*

  set the symbol size to *exp*, which must evaluate to a number. The symbol sizes have the same scale as font sizes.

- **fill** *fillstylecommands*

  use *fillstylecommands* for the symbol.

- **line** *linestylecommands*

  use *linestylecommands* for the rim

Example:

```
g1.gr1.s1 symbol style diamond
symbol size 18
symbol fill  on
symbol fill color 0xff0000
symbol line color 0x0000ff
symbol line width 2
```

Top

Kill commands

- **kill**[ ]**all**

  kill everything (clean slate) and initialize graph 1 in default state. The space between the keyword and the number is optional.

- **kill** (*graphdesc*|*groupdesc*|*setdesc*)

  remove graph/group/set. If the last graph was killed graph 1 is initialized in a default state. Example:

```
kill g1.gr12
```

Top

Copy/Move commands Use these to move sets around or copy them.

- (**copy**|**move**) *setdesc1* **to** *setdesc2*

  copy/move set1 to set2. Moving is essentially renaming. Note, that you can move a set to a different graph and/or group

```
# here I shall have a g1.gr1.s1
g2 on
# now we have graph 2
gr3 on
# now we have a group3 in graph 2
# in order to move g1.gr1.s1 there, we have to explicitly spell out g1.gr1
# since the last command defined the current descriptor as g2.gr3, hence:
move g1.gr1.s1 to g2.gr3.s11
# now we have moved the set into s11 in graph 2 group 3
# in order to move the s11 to s1 in g2.gr3 we do not have to change groups or graphs
# hence,
move s11 to s1
# will work
```

Top

Hook commands Sometimes it is usefull to have the program do something when the mouse is clicked on a certain point in a graph. This can be done by "hooking" a particular command to the mouseclick.

- **hook mouseclick left** *command-as-string*

  set the command in *command-as-string* to be hooked onto the left mouse click. In the moment only left mouse click is implemented. When hooking is active the current world point (at clicking) is written into the file `+currentpoint`. *command-as-string* can stretch over multiple lines to allow for several commands. Example

```
hook mouseclick left "
 with g1.gr10
  s1 on
```

```
  s1 length 2
  x[0]=cursor.x
  x[1]=cursor.x
  y[0]=world.ymin
  y[1]=world.ymax
  line color 0xff0000
"
```

This will draw a vertical red line at the mouse position when clicked.

Top

Cursor reference If **hook** commands are active the world coordinates pointed to by the mouse cursor can be references.

- **cursor** . (**x**|**y**)

  reference the world coorinate under the mouse cursor. Example

  ```
  hook mouseclick left "world xmin cursor.x
  world ymin cursor.y"
  ```

  Note, the newline within the string!

Top

Assignments/Definitions One can in a limited way define variables: scalars and vectors.

- *identifier* = *exp*

  define a variable name *identifier* and assign it *exp*. Note, that "length" cannot be an *identifier* name. Example:

  ```
  N=100
  m=mesh(0.1,12.3,N)
  ```

- [*setdesc* .](**x**|**y**) [*exp1*] = *exp2*

  this assigns the *exp2* to a value x[*exp1*] or y[*exp1*] in the current or specified set. Note, that the square brackets are for real. It is a vector element reference. The vector assignment is explained here. Example:

  ```
  s1.y[0]=0
  s1.y[length-1]=0
  ```

Top

Expressions *exp* can be anything of the following:

- **world** . (**xmin**|**xmax**|**ymin**|**ymax**)

  reference to the current world limits. (For hook commands and other usefull stuff.)

- **mesh** (*exp1* , *exp2* , *exp3*)

  define a vector of lenght *exp3* made of equidistant points between *exp1* and *exp2*. Example:

```
s1 on
N=100
s1 length N
x=mesh(0.1,12.3,N)
y=x^2
```

- *real*

  any real or integer number.

- *identifier*

  any previously defined *identifier*

- *parameter*

  any parameter defined on the command line. See *parameter*.

- **cursor** . (**x**|**y**)

  a reference to the mouse cursor position. (For hook commands) See here.

- [*setdesc* .] **length**

  the length of the set's x/y-vectors. Example.

```
with s1
x[length-1]=100
```

- [*setdesc* .] (**x**|**y**)

  the set's x/y-vector. Example.

- [*groupdesc* .] (**x**|**y**)

  the group's x/y-vectors

- *setdesc* . (**x**|**y**) [*exp*]

  the *exp*-th element of the x/y-vector of set. The square brackets are for real here. Example:

```
s1.x[10]
```

- **moment** (*setdesc* , *exp1* , *exp2* , *exp3*)

  the *exp1*-th moment of set.y in the x-interval [*exp2*,*exp3*]. These are normalized moment $M_n = \frac{\int_{x_0}^{x_1} f(u)u^n du}{\int_{x_0}^{x_1} f(u)du}$, hence $M_1 \equiv 1$. The second moment corrected for the center of gravity is defined as $\frac{\langle (x-M_1)^2 \rangle}{\langle\rangle} = M_2 - M_1^2$. Example:

```
killall
gr1 on
N=1000
a=0.5
x0=-0.6
x1=1.6
w=0.2
s1 on
s1 length N
x=mesh(x0,x1,N)
y=exp(-((x-a)/w)^2/2)*3
m1=moment(s1,1,x0,x1)
m2=moment(s1,2,x0,x1)
# calculate the normalized width
wi=sqrt(m2-m1^2)
echo "wi=",wi
gr1 irregular tics on
itic1 type x major
itic1 position m1
itic1 length 1
itic1 label side opposite
itic1 label "m1"
itic2 type x major
itic2 position m1+wi
itic2 length 1
itic2 label side opposite
itic2 label "wi"
itic2 line style dash
autoscale
```

- (*exp*)

  we can use parentheses around any expression. Example:

```
a*(b+c)
```

- *function* (*exp*)

function	meaning	*function*	meaning	*function*	meaning
sqrt(x)	$\sqrt{x}$	sin(x)	$\sin(x)$	asin(x)	$\arcsin(x)$
abs(x)	$\lvert x\rvert$	cos(x)	$\cos(x)$	acos(x)	$\arccos(x)$
exp(x)	e^x	tan(x)	$\tan(x)$	atan(x)	$\arctan(x) \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
log(x)	$\log_e(x)$	cot(x)	$\frac{1}{\tan(x)}$		
log10(x)	$\log_{10}(x)$			theta(x)	$\theta(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$

- *function* (*exp* , *exp*)

function	meaning
min(x1,x2)	$\min(x_1, x_2)$
max(x1,x2)	$\max(x_1, x_2)$
sign(x1,x2)	$\lvert x_1\rvert \operatorname{sign}(x_2)$
atan2(x1,x2)	$\arctan\left(\frac{x_2}{x_1}\right) \in [-\pi, \pi]$

- *exp1* ^ *exp2*

  exp1 raised to the power of *exp2*.

- *exp1* (*|/) *exp2*

- (+*exp*|-*exp*)

- *exp* (+|-) *exp*

Top

4.5.2 GUI

4.5.2.1 Plotting window

Use the right mouse key on objects and double click (depends on where you do this). Check out the mouse button tips (Fig. 4.2).

If the zoom buttons on the left are used, some of them change the cursor into a corss hair shape. This means that the corresponding function is on. To cancel the function use right mouse click.

Hotkeys:

zoom in	Ctrl-+
zoom out	Ctrl- -
autoscale all	Ctrl-a
autoscale x	Ctrl-x
autoscale y	Ctrl-y

scroll Ctrl-left, Ctrl-right, Ctrl-up, Ctrl-down

Hover with the mouse pointer over GUI elements to get tooltips.

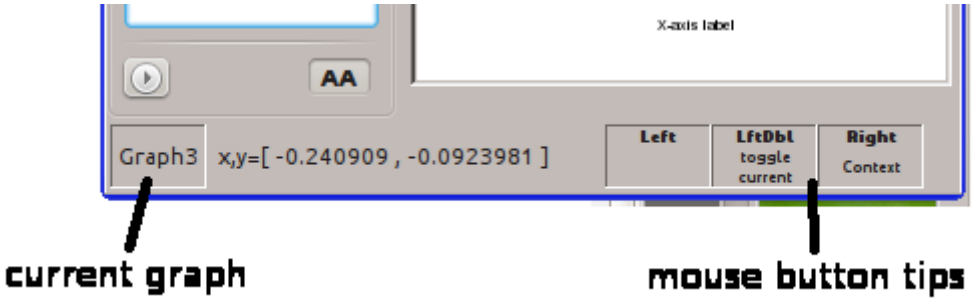


Figure 4.2: The botom of the main window with current graph indicator and mouse button tips.

To make a graph the current graph double click on empty space in the graph far enough away from sets and other objects. If several graphs overlap where you double click the current graph is changeing everytime you double click (loop through all graphs under the mouse cursor). You can also double click on empty space outside of any graph to loop through all the graphs. The current graph is displayed at the buttom of the window (Fig. 4.2).

4.5.2.2 *Scripting window*

Start search via Ctrl-F, type the search term, use backspace for corrections. Hit Ctrl-F to find next hit. If end of script is reached (colored search bar) hit Ctrl-F again to go back to start searching at the beginning of script. Use any cursor movement to cancel search mode. If Ctrl-F is hit to initiate search, hit it again to bring back the previous search string (if it exists).

4.5.3 *Logarithmic plots*

In log scale invalid data points are droped from all sets before plotting.

For ellipses in world coordinate units the radii for y/x direction denote the upper/right radius in world scale when the ellipse were placed at 1,1. Example:

```
killall
g1 on
x axis scaling log
y axis scaling log
world x 0.1 , 100
world y  0.1 , 100
```

```
Ellipse1 on
Ellipse1 name ""
Ellipse1 line style solid
Ellipse1 line width 1
Ellipse1 line color 0x0
Ellipse1 fill on
Ellipse1 fill color 0xeeeeee
Ellipse1 coordinate system World
Ellipse1 center 1,1
Ellipse1 radii 9,9
Ellipse1 angle  0
with g1.gr1
s1 on
data xy
10 0.1
10 100
end data
s2 on
data xy
0.1 10
100 10
end data
textbox1 on
textbox1 coordinate system World
textbox1 "radius 9"
textbox1 position 11 , 1
textbox2 on
textbox2 coordinate system World
textbox2 "center at 1,1"
textbox2 position 11 , 9
Line1 on
Line1 coordinate system World
Line1 start  11.2658,5.96763
Line1 end  1.01985,0.796076
```

4.5.4 Text formating

There is a rudimentary amount of text formating for textboxes:

- $~
  next character is from the symbol font (greek) (hopfully works on your system)
- $i

switch to italic font

- $n

  switch to regular font

- $_

  switch to next subscript level

- $^

  switch to next superscript level

- $.

  switch to normal level

- $x{real-number}

  shift the current position (for the following characters) to the right (positive number) or left (negative number). For this to tak effect in sub/super scripts, the $x command must preceed $^ or $_. Example: try

  ```
  X$_ij$.$x{-0.5}$^ab$.
  ```

  or

  ```
  X$_a$_i$.$x{-0.8}$^$~m$.
  ```

  versus

  ```
  X$_ij$.$^ab$.
  ```

  or

  ```
  X$_a$_i$.$^$~m$.
  ```

- $arrowup, $arrowdown, $arrowleft, $arrowright, $angstroem, $infinity

  clear?

- $$

  a $ sign

Note: that some special characters (e.g. $arrowup) switch the font to regular after they were printed (bug). To continue with italic, use another $i. On some systems the on-screen display of symbol characters is wrong, but the printed characters are correct.

4.5.5 Data

The data are organized in groups and sets. Groups and sets have attributes. The group attributes can be set to hold for each set of the group irrespective of the individual set's settings.

4.5.6 Files

XFBP saves files in its own format (which is a subset of the scripting language Sec. Scripting). The extension is `.xfp`. It can also save and load the scripts, usually with the extension `.cmd`. It can load a set of data files, see Sec. Data file types.

4.5.7 Command line options

The program can be called with filenames as argument. If the file type has to be specified a file type flag has to preceed the filename. Most common files are automatically recognized via some heuristics. These are FPLO band and bandweight files and XFBP's own files `.cmd` `.xfp` files. If no file type is specified data files are browsed with the `-xny` flag assumed (blocks of columnar data). In general each file is read into its own group. Spin polarized band structures create two groups if there are two spin directions (not full relativistic).

`-cwd:` make directory of following file current directory in the application. (I do not know, what this was for.)

`-oi:` open in observe mode, to monitor calculation progress (especially usefull for an on-machine FPLO run).

`-a:` specify a parameter-name value pair, e.g. `-a p1:"filename1"`, This can be used in generic command files, where `the parameter $p1` will be available in the script and contain "`filename1`".

4.5.7.1 File type flags

For more details on the data files see Sec. Data file types.

`-xny:` xny data

`-xynw:` xnyw data

`-xynz:` xynz gridded data

`-band:` band structure data

`-bandweight:` band weights data

-akbl: Bloch-Spectral-Density data

-p: a parameter file, containing scripting commands.

4.5.8 Data file types

xny: Data sets are read, assuming that an empty line starts a new data block. In each multi column block the first column is x and the other columns are y_i, resulting in as $N_{\text{column}}-1$ sets for each data block in the file. All sets end up in a new group.

xynw: The first column of each block is x. The second is y and the following columns are weights.

xynz: (Experimental) First comes a block of x values, each line one. Then comes a similar block of y values. Finally a block of z values. The resulting plot will be a density plot where the z-values define the gray scale.

band: An FPLO band structure file.

bandweight(s): An FPLO band weights file.

akbl: An FPLO Bloch-Spectral-Density file. (CPA, FPLO5)

4.6 Occupation matrix manipulator: DMATEDIT

LSDA+U uses onsite occupation number matrices for certain $nl-$shells (defined by the user in FEDIT). These matrices are saved in `=.dmat_init` and serve as input for consecutive runs. Sometimes the user needs to see/edit them in some local axes and/or complex harmonics basis in order to prepare starting conditions for new runs in order to force the calculation into one of the many possible local (meta) stable solutions of LSDA+U. This is provided by this program. It needs the file `=.dmat_init`, which gets created once LSDA+U was switched on and FPLO was runing at least once after switch on. It reads this file, lets you manipulate it and saves it back on Save/Quit. **Note, that the program always saves the file `=.dmat_init` in real harmonics basis of in the global cartesian coordinate system. It uses transfomrations defined by all the settings in the dialog to display the matrices appropriately in local axis and chosen harmonics, but does save the matrices after transforming them back into the global system.** That means the file can look different from what's shown on screen. In full-relativistic calculations the spin density matrix $n^{s's}_{mn}$ can be non-diagonal. It is defind within global real harmonics but in the spin frame corresponding to the xc-field quantization axis specified in the FEDIT main menu. This means that its off-diagonal part $n^{\uparrow\downarrow}_{mn}$ should be small. If this is not the case, especially if the diagonal of $n^{\uparrow\downarrow}_{nn}$ is sizeable, the colinear aproximation of the full-relativistic mode is violated by LSDA+U. The off-diagonal

part (denoted "Spin mixed") is only shown if present and only the up-down block. Note, that the occupation numbers define the shell occupation, shell spin moment and shell L_z-moment.

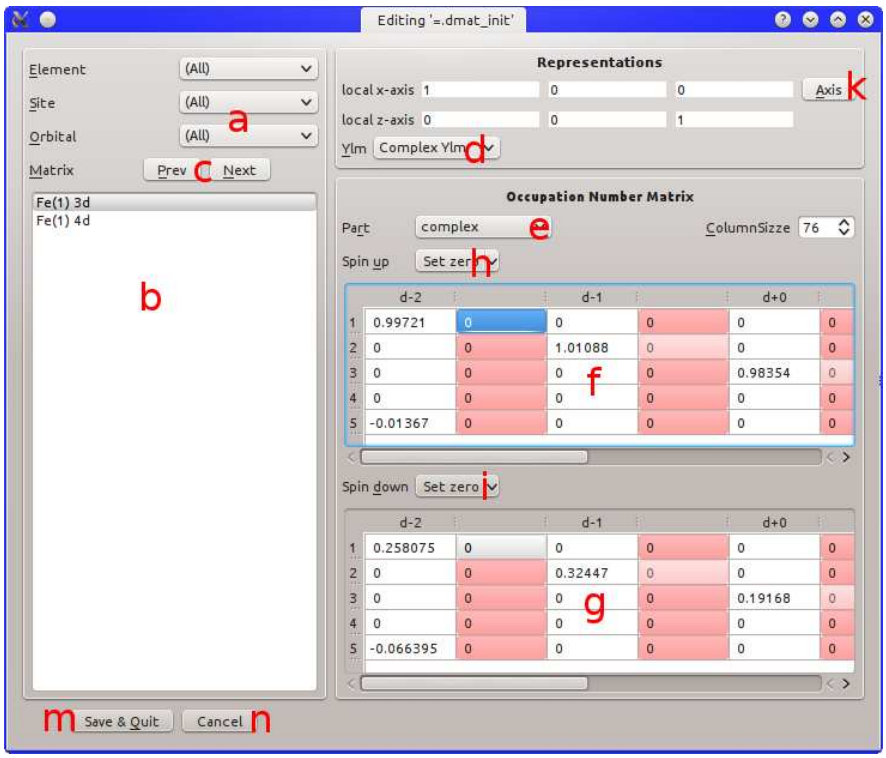


Figure 4.3: DMATEDIT for non-full-relativistic case

a) Filters defining, which matrices are shown in the occupation matrix list (b) and can be browsed through via the Prev and Next buttons (c). Only the shells for which LSDA+U was actually enabled are shown in this list.

b) Occupation matrix list filtered by (a)

c) these buttons browse through the matrices shown in the list (b).

d) Switch between real and complex spherical harmonics as basis for the matrices.

e) Chose to show the real, imaginary or both parts in the matrices (f,g)

f,g) Spin up and down occupation number matrix for the selected shell. Here you can edit the matrices!

h,i) Set default values. This is a tool button. The arrow to the right allows to select which default it sets. Clicking on the button part sets the currently selected default

k) Change the local axis in which the harmonics are defined. This is needed for instance if the local environment of the atom is rotated versus the global cartesian system.

m) The matrices are saved back to `=.dmat_init` and the program exits.

n) (Hotkey <Escape>) quit the program without saving.

DMATEDIT saves local axes and such information in `dmatedit.ini` in the current directory.

4.7 Optics data: FOPTICS

Optical functions are basically variations of the energy dependend $\boldsymbol{q} = \mathbf{0}$ dielectric tensor $\varepsilon_{ij}(\omega)$. It consists of an intra- and an inter-band part. The intra-band part is usually taken from the Drude model, since scattering processe, which lead to a finite life-time Γ of the electrons are not included in standard bandstructure calculations.

The Drude model contains the plasma energy tensor ω^P_{ij}, which can be diagonalized resulting in principal axes along which the plasmon has energy $\omega^P_{1,2,3}$ and a life-time $\Gamma_{1,2,3}$. The life-times are user input parameter and the plasmon energies can be chosen by the user as well. However, the plasmon energies and principal axes also get calculated by FPLO and are written to the output and into the file `+plasmon`. Note, that in spin-polarized calculations the principal axes can actually differ between different spin directions (spin up, spin down and total spin sum), which is why all three cases have their own plasmon information in `+plasmon`.

The inter-band part is more expensive to calculated and depends on the actual band structure and on optical matrix elements. The implementation actually only calculates $\mathrm{Im}\varepsilon(\omega)$ for $\omega \geq 0$, since $\mathrm{Re}\varepsilon(\omega)$ can be obtained via the Kramer-Kronig relation. FPLO calculates the optical matrix elements $\boldsymbol{p}$ and performs the k-integration in the expression

$$\mathrm{Im}\varepsilon^{\mathrm{inter}}_{ij}(\omega) \quad = \quad -\frac{1}{\omega^2}\frac{4\pi^2}{V}\sum_{\boldsymbol{k},\overline{n},n}{}' \left[f(\varepsilon_{\boldsymbol{k},n}) - f(\varepsilon_{\boldsymbol{k},\overline{n}})\right]\delta(\varepsilon_{\boldsymbol{k},n} - \varepsilon_{\boldsymbol{k},\overline{n}} - \omega)\, \boldsymbol{p}_{i\boldsymbol{k}n\overline{n}} \circ \boldsymbol{p}^*_{j\boldsymbol{k}n\overline{n}}$$

and writes the result into the file `+imeps`. This file is not meant to be used directly, but one can look at it via XFBP for convergence controll. You can control this process via the `OPTICS` submenu in FEDIT (read the help screen).

Caveats: The inter-band k-integral is not very smoothly converging especially if many small Fermi surfaces are present. Always check k-point convergence of `+imeps`. Furthermore, FPLO is a small basis method, which means that there are not very many unoccupied bands, which means that optics gets less and less accurate the higher the energy ω is.

After FPLO has created the file `+imeps` some optical functions can be created in a separate step, in which also the intra-band part gets added. The program FOPTICS is a command line tool. It has command line options, which are avaialble via FOPTICS `-h`:

```
usage:
```

```
foptics10.00-44-x86_64 [-i fplooutput_imag_part_interband_eps] ...
[-os suffix_for_output_files] [-h]  ...
[-gamma gamma1 [gamma2 ... gamman]] ...
[-omega omega1 [omega2 ... omegan]] ...
[-skip nskip] [-nointer] [-nointra] [-ni]

fplooutput_imag_part_interband_eps: normally +imeps
omega: the plasmon frequencies for the intraband contributions in eV
gamma: the broadening (life time) for the intraband contributions in eV
nskip: take only every nskip data point
-nointer: only take the intraband contributions
-nointra: only take the interband contributions
-ni = not interactively: take values from +plasmon and from -gamma and -omega
but do not ask for not explicitly specified values.

There are 3 gamma values: for each principal axis one.
There are 3 omega values along the principal axes for each spin direction
and for the total (spin sum). This is so, since the principal axes
need no coincide for spin up, spin down and the spin sum.
In non-spin-polarized cases ony spin1 is used and equals the spin sum.


This program reads inter-band output from FPLO (+imeps)
and calculates some optical functions. The user can define
plasmon frequencies and broadenings to include the intra-band
contributions.

The program also reads the file +plasmon to extract the calculated
plasmon energies and the principal axes of the plasmon tensor.
The user can overwrite the plasmon energies by interactive input
or by the option -omega.

The output functions are
re_eps: real part of epsilon
im_eps: imaginary part of epsilon
re_sigma: real part of the optical conductivity in inverse Ohm*cm
loss: the loss function
For the individual spin contributions _spin1/_spin2 is appended to the
file names. (Only for spin polarized cases.)
If an output suffix (-os) is defined it will be appended to all file names.
```

One can also just call the program without options. It reads `+imeps` to obtain $\mathrm{Im}\varepsilon^{\mathrm{inter}}(\omega)$ and `+plasmon` to obtain the eigen decomposition of $\underline{\omega}^P = \sum_i \boldsymbol{Z}_i \omega_i^P \boldsymbol{Z}^T$, where $\boldsymbol{Z}_i$ is a principal

axis of the tensor. The program writes the plasmon and life-time information to standard out and asks the user for input of ω_i^P and Γ_i unless option `-nointra` or `-ni` was set or values were provided on the command line. The command line option `-gamma` reads up to 3 Γ_i, one for each principle axis. These Γ_i are used for all spin directions. The command line option `-omega` reads up to 9 ω_i^P, the first three are $\omega_{1,2,3}^{P\uparrow}$, the next three are $\omega_i^{P\downarrow}$and the last three are $\omega_i^{P(\uparrow+\downarrow)}$. If not all $\omega_i^P(\Gamma)$ values are provided the program will interactively ask for user input for the non-specified values. The default ω_i^P values are taken from `+plasmon`. If option `-ni` (not interactively) is set, all values specified via `-omega` and `-gamma` are used and the not explicitly specified values are taken form `+plasmon`. So, interactivity can be avoided by either specifying all values via the options (3 Γ and 3ω (non-polarized) or 9ω spin-polarized) or by using `-ni`, which is usefull for falling back to the default values from `+plasmon` without having to type enter for all questions asked. Play with it to get the idea.

FOPTICS will use the Kramers-Kronig transformation to calculate $\mathrm{Re}\underline{\varepsilon}^{\mathrm{inter}}$ from $\mathrm{Im}\underline{\varepsilon}^{\mathrm{inter}}$. Then the intra-band part Eq. (4.1) is determined for which the plasmon tensor is constructed from the principal axis $\boldsymbol{Z}_i$ and the (user-specified) paramters Γ_i and ω_i^P. Finally,

$$\underline{\varepsilon} = \underline{\varepsilon}^{\mathrm{intra}} + \underline{\varepsilon}^{\mathrm{inter}}$$

is calculated and the optical functions are derived form it and written into separate files. Note, that the two parts can be switched off via the options `-nointer` and `-nointra`. Currently, there are the following optical functions (see Sec. 4.7.1)

function	file name
$\mathrm{Re}\varepsilon_{ij}\,(\omega)$	re_eps[_spin1|_spin2]
$\mathrm{Im}\varepsilon_{ij}\,(\omega)$	im_eps[_spin1|_spin2]
$\mathrm{Re}\sigma_{ij}\,(\omega)$	re_sigma[_spin1|_spin2]
$L_{ij}\,(\omega)$	loss[_spin1|_spin2]

Table 4.5: Optical functions

These files contain comments for user orientation and they can most conveniently be displayed with XFBP. Note, that only the non-zero tensor elements are written. A symmetry analysis of the inter-band part is used to determine these tensor elements. In this it is assumed that the intra-band part must have the same symmetry.

The options `-i` and `-os` are usefull if several different settings (e.g. k-points) are used and the file `+imeps` was renamed into e.g. `imeps_12_12_6` and `imeps_42_42_12` for the different runs by the user. Then FOPTICS `-i imeps_12_12_6 -os 12_12_6` reads `imeps_12_12_6` and produces `re_eps_spin1_12_12_6` and so on.

Example Al We make a standard calcultion for Al as explained in getting started. Then we go to the `OPTICS` submenu of FEDIT and switch on optics and set the upper energy bound to 20 eV (this is for demonstration of the plasmon peaks, the interband part is not really accurate at such high energies!). We re-run FPLO. The UNIX command

```
ls -ltr
```

shows that the file +imeps got created. For orientation open it via

```
Xfbp +imeps
```

Copy it:

```
cp ./+imeps ./+imeps_12
```

(to indicate the default $12 \times 12 \times 12$ k-mesh). Change the k-mesh in Fedit to $42 \times 42 \times 42$. Re-run Fplo. (This will require a few steps to achieve the self-consistency for this k-mesh.) Copy it:

```
cp ./+imeps ./+imeps_42.
```

Load both files

```
Xfbp +imeps_12 +imeps_42
```

You see the convergence issue. Make another calculation for a $60 \times 60 \times 60$ k-mesh, copy the file and compare the +imeps files. (Aluminum really has tiny Fermi surfaces.)

Now, run

```
Foptics -i +imeps_60 -os 60
```

and hit enter for all questions or use option -ni to take the default values. Open the resulting loss function

```
Xfbp loss_60
```

(the suffix _60 came from the -os 60 option). Compare the position of the plasmon peak at 15eV with the bare plasmon frequency 12.3eV in the Foptics output. This shift is due to the inter-band part. Now, calculate the intra-band only loss function:

```
Foptics -i +imeps_60 -os 60_intra -nointer
```

Now, the plasmon peak is at the bare energy 12.3eV. At last calculate the inter-band only loss function:

```
Foptics -i +imeps_60 -os 60_inter -nointra
```

(Beware of the correct placement of inter and intra!!) Compare the three:

XFBP `loss_60 loss_60_intra loss_60_inter`

You must see something like Figure 4.4.

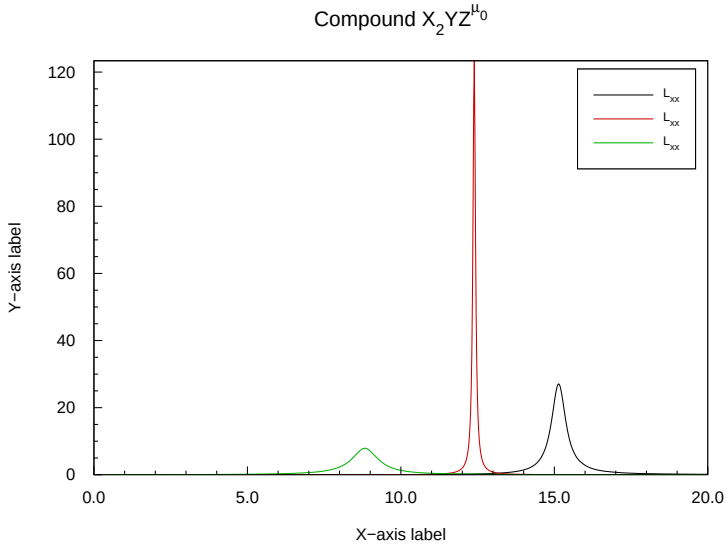


Figure 4.4: Total, intra-band and inter-band loss function for Al.

This example is also discussed by the Wien2k implementation. Finally, you can have a look at all the other functions (Note that the intra-band part of ε diverges at $\omega = 0$).

4.7.1 Optical functions

4.7.1.1 General

The immediate function is the optical dielectric function (without momentum transfer)

$$\boxed{\varepsilon(\omega) = \varepsilon^{\text{intra-band}}(\omega) + \varepsilon^{\text{inter-band}}(\omega)}$$

where the intra-band expression must be approximated via a Drude model and the interband part is the expensive thing the optics-module is all about. ε is connected to the optical conductivity via

$$\sigma(\omega) = -i\frac{1}{4\pi}\omega(\varepsilon(\omega) - \mathbf{1})$$

which makes σ a measure of the imaginary part of ε.

$$\boxed{\text{Re}\sigma(\omega) = \frac{1}{4\pi}\omega\text{Im}\varepsilon(\omega)}$$

$$\mathrm{Im}\sigma\left(\omega\right)=\frac{1}{4\pi}\omega\left(\mathbf{1}-\mathrm{Re}\varepsilon\left(\omega\right)\right)$$

The Loss function is defined as

$$\boxed{L_{\alpha\beta}=-\mathrm{Im}\left(\frac{1}{\varepsilon\left(\omega\right)}\right)_{\alpha\beta}}$$

4.7.1.2 Intra-band (Drude)

The intraband contribution is approximated by

$$\boxed{\underline{\varepsilon}^{\mathrm{intra}}\left(\omega\right)=\mathbf{1}-\frac{\underline{\omega}_P^2}{\omega\left(\omega+i\Gamma\right)}} \tag{4.1}$$

which gives

$$\boxed{\mathrm{Re}\underline{\varepsilon}^{\mathrm{intra}}\left(\omega\right)=\mathbf{1}-\frac{\underline{\omega}_P^2}{\omega^2+\Gamma^2}}$$

with a root at

$$\omega_0=\sqrt{\underline{\omega}_P^2-\Gamma^2}$$

and

$$\mathrm{Im}\underline{\varepsilon}^{\mathrm{intra}}\left(\omega\right)=\frac{\underline{\omega}_P^2\Gamma}{\omega\left(\omega^2+\Gamma^2\right)}$$

which results in

$$\boxed{\mathrm{Re}\sigma^{\mathrm{intra}}\left(\omega\right)=\frac{1}{4\pi}\frac{\underline{\omega}_P^2\Gamma}{\omega^2+\Gamma^2}}$$

$$\mathrm{Im}\sigma^{\mathrm{intra}}\left(\omega\right)=\frac{\omega}{\Gamma}\mathrm{Re}\sigma^{\mathrm{intra}}\left(\omega\right)$$

For the loss function we get (in the diagonal principal axis frame of $\underline{\omega}^P$)

$$L_{\alpha\alpha} \quad = \quad \omega\Gamma\frac{\omega_{P\alpha}^2}{\left(\omega^2-\omega_{P\alpha}^2\right)^2+\omega^2\Gamma^2} \tag{4.2}$$

with the maximum at

$$\begin{aligned}\omega_{\mathrm{peak}} &= \frac{1}{\sqrt{6}}\omega_{P\alpha}\sqrt{2-\frac{\Gamma^2}{\omega_{P\alpha}^2}+\sqrt{16-4\frac{\Gamma^2}{\omega_{P\alpha}^2}+\frac{\Gamma^4}{\omega_{P\alpha}^4}}}\\ \omega_{\mathrm{peak}} &\approx \omega_{P\alpha}\left(1-\frac{1}{8}\frac{\Gamma^2}{\omega_{P\alpha}^2}\right)\end{aligned}$$

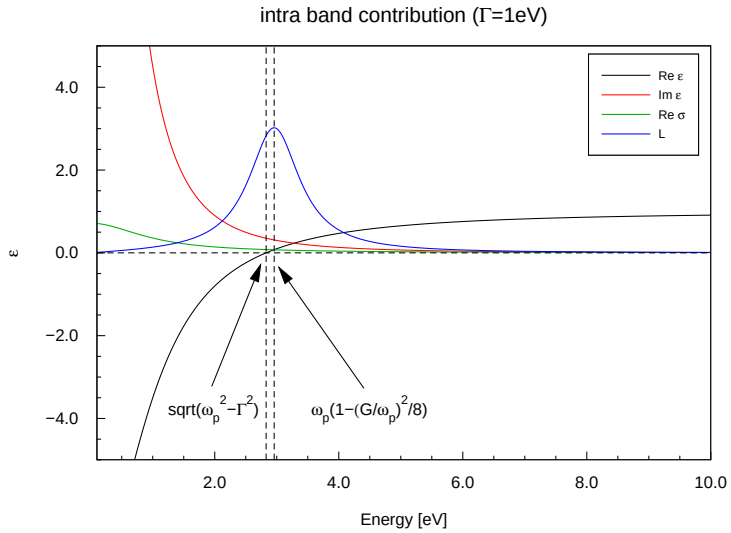


Figure 4.5: Intra band contribution for an unaturally large $\Gamma = 1\text{eV}$ (to resolve the small shifts of ω_P.)

4.7.1.3 *Intra-band and constant interband* ε_∞

Now, we include a constant approximation for the interband contribution ε_∞.

$$\begin{aligned} \underline{\varepsilon}(\omega) &= \mathbf{1} - \frac{\underline{\omega}_P^2}{\omega(\omega + i\Gamma)} + \varepsilon_\infty \\ &= (1+\varepsilon_\infty)\left(\mathbf{1} - \frac{\underline{\tilde{\omega}}_P^2}{\omega(\omega+i\Gamma)}\right) \end{aligned}$$

where we defined the renormalized plasmon frequency as

$$\tilde{\omega}_P = \frac{\omega_P}{\sqrt{1+\varepsilon_\infty}}$$

This means that in all expression we have to replace $\omega_{P\alpha} \to \tilde{\omega}_{P\alpha}$ and devide (multiply) by $\frac{1}{1+\varepsilon_\infty}$.
Eq. (4.2) then gives

$$L_{\alpha\alpha} = \frac{1}{1+\varepsilon_\infty}\omega\Gamma\frac{\tilde{\omega}_{P\alpha}^2}{\left(\omega^2 - \tilde{\omega}_{P\alpha}^2\right)^2 + \omega^2\Gamma^2}$$

with the peak at

$$\boxed{\omega_{\text{peak}}^{\text{intra}+\varepsilon_\infty} = \tilde{\omega}_{P\alpha}\left(1 - \frac{1}{8}\frac{\Gamma^2}{\tilde{\omega}_{P\alpha}^2}\right)}$$

which is better suited to determine the plasmon peak than extracting the divergent Drude part from σ.

Chapter 5

Automation, scripting, pipe-mode

The manipulation of input files via unix commands like `ed`, `sed`, `awk` and similar ones is strongly discouraged. The format of the input files follows a syntax and is not fixed in position. To achieve automation the pipe-mode of FEDIT can be used.

5.1 Rules

There are only a few rules to be obeyed.

1. The menus/screens/edit-controls of FEDIT are operated by ascii hotkeys in interactive mode. For some actions there are control keys (cursor movement, searching and scrolling). In pipe mode the latter keys are not needed at all. The ascii hotkeys are replaced by a simple syntax explained below.
2. In interactive mode sometimes only a portion of the menu's form is seen on screen. In pipe-mode allways the full form is virtually visible. There is no need for scrolling. (In fact there is no need for scrolling in interactive mode as well, since the currently invisible hotkeys are also active and just typing an currently invisible hotkey will scroll the form to make the selected edit-control visible.)
3. In interactive mode the edit-controls may be edited to change only part of the controls content. In pipe-mode the full content/data of the edit-control has to be entered.
4. In interactive mode there are toggling actions, in pipe mode these become normal edit-controls.
5. In interactive mode there will be informational screens displayed after certain actions, which just show output. In pipe-mode these screens will not occure. (E.g. the FPLO message after symmetry update.)
6. In interactive mode there will be questiones to be answered depending on the context. In pipe-mode such questions naturally cannot occure.

7. In interactive mode, the user enters information by typing keys, in pipe-mode the user input to FEDIT is read from standard input (`stdin`). This input may e.g. be stored in a file or come from a `here-script` within a shell/perl-script.

8. In interactive mode the user feedback is what is seen on the screens, in pipe-mode the feedback is written to standard error (`stderr`). So it is good practice to redirect `stderr` to a file and to redirect standard output to `/dev/null`. (The screens on `stdout` will change so rapidly that they are useless.) Let us assume that the input for the pipe-mode is stored in the file `=.pipe`. (The prefix indicates that it is an essential input file.) A good way to feed the information contained in `=.pipe` into FEDIT in a shell environment would be

   ```
   cat ./=.pipe | FEDIT  -pipe 2>./+log 1>/dev/null
   ```

   or

   ```
   FEDIT  -pipe < ./=.pipe 2>./+log 1>/dev/null
   ```

   Both commands have the same effect.

   (a) The first command uses `cat` to write the content of `=.pipe` to `stdout`, which than is redirected to the `stdin` of FEDIT via the unix pipe command |. (That is the origin of the name pipe-mode.) The second command uses the unix tool for redirecting the `stdin`. Here the content of `=.pipe` is directly written to the `stdin` of FEDIT.

   (b) The editor is given the option `-pipe` to setup the mode.

   (c) The `stderr` is redirected to the file `./+log`. (The naming is up to the user, however, our choice follows the rules explained in Chapter 3. The '`+`' indicates that the file is not essential and may be deleted after a succesful run.)

   (d) The `stdout` is redirected to the unix device `/dev/null`, which just means to discard it. (`/dev/null` is a 100% information sink :-)

9. The return code of FEDIT should be checked and the script aborted if needed, to intercept input errors. FEDIT sets the shell exit/return code as

`1`	on success
other	on error

   The exit/return code must be checked immediately after the command, which returned it. Any shell action in between changes the exit/return code!

10. The logic of the pipe-mode is such that the user has to feed hotkey-data sequences and menu-hotkey sequences into FEDIT as he would in interactive mode. The user creates key sequences in the pipe-input, which navigate through the menus as if it would be an interactive session. This includes the `x`-hotkey to leave a sub menu. It excludes the `x`-hotkeys of information screens as described above. (The main reason is, that those

screens are context dependend.) If there is any invalid input in the pipe file, the editor will abort unsuccesfully. The reason for the failure is printed to `stderr`, which is accesible as explained above. To create pipe input, just use the editor interactively and write down or remember all hotkeys, which are pressed to complete the desired editing (except the information screens and questions).

Altogether the simple rule is that in pipe-mode there is only navigation to certain positions and entering of data. Every other user ⇔ FEDIT interaction will be absent.

5.2 Syntax

We assume that the pipe input is stored in a file. Its syntax is as follows.

1. The file may contain comments, which are lines whose first non-blank character is '`#`'.
2. The file may contain empty/blank lines.
3. Every single action goes into a separate line.
4. A hotkey is wrapped into two `'@'` characters. (If `perl` is used for the input creation the character '`@`' has a special meaning in `perl` and must be protected '`\@`'.)
5. An alternate sub menu ('<SPACE>hotkey') is called with a single space followed by the hotkey, wrapped into two '`@`' characters
   ```
   @ hotkey@
   ```
6. A hotkey sequence, which activates an edit-control to enter data is mapped onto a line
   ```
   @hotkey@data
   ```
7. A hotkey sequence, which activates N edit-controls is mapped onto
   ```
   @hotkey@data1@data2@...@dataN
   ```
   Example: The symmetry menu contains the Wyckoff position definitions, which consist of an element name and a vector. The hotkey equals the sort number. A corresponding pipe input to edit the second sort could look like
   ```
   @2@ Co @ 2/3 1/4 ,
   ```
   This sets the Wyckoff position definition for sort 2. The comma ',' is a duplicator of the value preceeding it[1]. Thus
   ```
   1/4  , 2/3 = 1/4 1/4 2/3
   1/4  ,,    = 1/4 , , = 1/4 1/4 1/4
   ```
8. An edit-control may be activated by a search command (as in interactive mode). This is done by
   ```
   !search-string!data
   ```
   This will look for the first occurence of the search string in the current form and if this

[1] The comma also works in interactive mode.

markes an editable control, it is activated and data is used as the control's new data. Multiple edit-controls, selected by search are mapped onto

```
!search-string!data1@data2@...@dataN
```

(Note: As in the previous point, the '@' replaces the '<ENTER>' key used in interactive mode.) As in interactive mode, the parentheses enclosing the hotkeys on the FEDIT-screens are ignored in search mode. Example: the screen entry "Conver(G)ence condit" is searched for as "convergence condit".

9. An edit-control, which is a toggle in interactive mode is not toggling in pipe mode. So, if such data shall be edited in pipe mode one needs to set the value explictely. This assures a defined state of the input after completion.

   (a) **Old**: logical values may be 't' or 'f'. **Now**, the values are no longer differently represented on screen compared to options and can be set by 't', 'f' or '+', '-'.

   (b) binary values take there values as they are written on screen in interactive mode. (Example: spin sorts may be '1' or '2')

   (c) Options (marked as selected by [X] on screen) are selected or deselected with the values '+' or '-' in pipe mode

   Example:
   To select the option NO_SYMMETRYTEST in the options submenu we may use the search tool

   ```
   !NO_SYMMETRYTEST!+
   ```

   or to deselect it use

   ```
   !NO_SYMMETRYTEST!-
   ```

   The search string must be choosen to be unique in the way that it really selects what you want. It is best used only in options menus and in select boxes.
   It is strongly discouraged to activate options (the things marked with '[X]') via the related hotkey, since these hotkeys are created automatically by FEDIT and thus may change if the version changes[2]. The search utility is designed exactly for the purpose of changing options. In select boxes the hotkeys will not change. However, the search utility gives better readable pipe-files.

[2] In fact, there is a list of all possible options. FEDIT takes this list and creates hotkeys for all of them. If the order of options in this list changes between different FPLO versions, the mapping of hotkeys to options will naturally change as well. The authors try to avoid a change of the options order. However, there are thinkable cases when this will be nessecary!

10. The alternative choices in a select box are selected by a single hotkey command as in

```
...
# we assume we are in the main menu here
# we enter the relativistic select box (hotkey 'r' in the main menu)
@r@
# we select the full relativistic mode, by searching for
# 'full relat', which is uniqe in this select box
!full relat!
# We equally could use the hotkey, off course.
# Let us select it again:
@f@
# Now we leave the select box and go back to the main menu
@x@
# continue
....
```

Please observe that the parentheses indicating the hotkey ('(F)ull relativistic' in this case) are not visible in search mode, interactive or not.

11. The last key sequence in the pipe mode must be

```
@q@
```

and must be called being in the main menu. (So, you need to virtually navigate back to the main menu.) This is to assure proper cleanup.

5.3 How to set up automation

There is one pecularity. Suppose you created a series of input files and converged all related calculations. Now, you want to change some settings, for instance increase the number of k-points, since it turned out that you had to few of them. You could do this with a minimal script, which only changes the number of subdivisions in k-space. Later you recognize that you need to change other settings. Again you could do it with a minimal script. The main drawback of this approach is, that you loose control over the changes, which have been done. Another point is that after a change of certain symmetry parameters some input data are reset to default values. For these reasons it is advisible, to set up the pipe file(s) always such that the input files are completely determined by the content of the pipe file.

If FEDIT is called within a directory containing input files it offers the content of these files for editing. This means that the state of input files depends on the previous editing. The following sequence of actions will assure a state that does not depend on previous editing.

1. Go to the symmetry menu and enter all symmetry parameters explicitely. (These are not so many data, so it is not a big task.)

2. Call `update` with the '+' hotkey! This will reset certain data of the `=.in` (and `=.basis`) files. **Never forget this point!** Most of the settings will remain unaltered. All calcu-

lations of a series should belong to the same symmetry type. That means, the Wyckoff positions should at most change their parameters but the spacegroup should stay the same.

3. Go back to the main menu (hotkey '`x`')

4. Call the alternate submenu action `Recreate` (hotkey '<SPACE>`e`'). This will reset the file `=.in` to its default values, which belong to the symmetry setup in `=.sym`. Remember that in pipe mode the corresponding question (`Really rebuild default content ? (y/n):`) will not be asked, so you must not type '`y`' or '`n`' in pipe mode. Following the answer '`y`' an informational screen will appear in interactive mode, which will be left with '`x`'. This screen will be absent in pipe mode, due to the general rules above. That means that in interactive mode you would type the sequence '<SPACE>`eyx`' to perform the rebuild, but in pipe mode the action is achieved with a single line

```
# assume we are in main menu
# next line will call the reset action
@ e@
# we are back in main menu now, haveing default settings
```

5. Now, enter all input data, which differ from default input, except symmetry data.

A setup for a series of calculations may be done as follows.

1. Create interactively an initial input for your compound. Perform a self consistent calculation. Check if everything is fine. Converge the number of k-points (may already be done with the help of scripts, off course).

2. Set up the scripts, which create the input for the series. Make sure to perform every calculation in a seperate directory! After creating a new directory for a particular calculation of the series and before invoking FEDIT copy `=.sym` (`=.basis` for older FPLO versions) and `=.dens` from the initial calculation directory into the new directory. This has the advantage that the preconverged density will be available. As a result convergence will be much faster (in many cases) than in a calculation from scratch. This procedure makes sense as long as the parameter variation in the series is not to large. But even if so, the proposed procedure will not hurt.
   (In calculation series, (e.g. search of the minimum of an energy surface using e.g. a steepest decent algorithm) where the input paramters depend on previous calculations, one may copy one of the latest previous calculations into the new directory instead using the inital calculation.)
   Make sure that the '`=.`'-files are copied only, if a new directory has been created. (Do not overwrite converged basis or density.)[3]. If the script is re-run with some paramters changed, the density of the previous self consistent calculation should be retained!

[3] This means that the script, which creates the directories and the input should test the existence of the directory.

If other input files are used (e.g. `=.basdef`) make sure that they are copied too or that they are created by the script.

3. Run the script to create the directories with the proper input. Check the created input interactively, especially when new scripts are developed!

4. Launch all calculations in the various directories and wait for self consistence. Redirect output into a file, best is to allways use the same file name (for instance '`out`'). Check from time to time if everything runs fine.
   You may modify your script such that it serves the input creation and the starting of the calculations. Or you create a separate script to launch the calculations. Use the specifics of your platform (may be there is a job queue).

5. Check if all calculations realy converged. There is a final message on completion of a calculation[4]. Check this. The final message has the structure
   ```
   TERMINATION: Keyword : explanation
   ```
   where keyword may be

`Finished`	The calculation run to self consistence or a single step calculation finished.
`Normal`	The calculation did what it was ment to do and achieved the goal. Some actions terminate the program before self consistency, and this triggers a normal termination.
`Error`	Something happend. In many cases there is a cure for the problem.
`Crash`	This is really a bad case.

   The Keyword may be checked automatically with the help of unix tools. The explanation string gives a rough idea, what happend.

6. Have a look at example scripts in the distribution on how to extract certain data from the output.

7. Now, you are ready and may modify the script to change some settings and re-run the series if nessecary. Take care of all points mentioned so far.

5.4 Advanced features

5.4.1 Initial polarization, inital spin split

Some times, you may encounter a situation where it is nessecary to repeat an initial spin split for a series of calculations, since for example, the calculations were started spin polarized with an insufficient split and run back to a non magnetic state. Instead of discarding all calculations

[4] Except a bug occured.

done so far, it is better to force a new inital split on the pre-converged density. But, we face the situation that there are already 2 spin sorts in the density files and thus another `inital polarization` would urge FPLO to pose the question

```
Do you want to skip the possibly repeated initial spin splitting ?
```

Normally, the answer would be 'y' since mostly one just forgot to switch off the `inital polarization`. But in our situation the answer would be 'n', which forces another split. (What we really want is to restart the calculations in a new attractor basin.)

Now the problem is that usually the FPLO jobs are started in a background mode. In such a case there is no user to answer the question and so the job will crash (due to the fact that it tries to read from `stdin`, which is not present in background mode). If the job is not running in background it will hang and wait for the answer. Imagine, you started the script before weekend and come back at Monday and it still is waiting for the answer, or crashed.

There is a way to circumvent the problem: One provides FPLO with the nessecary input on `stdin`. If the program is not reading from `stdin` (since the question did not arise) the `stdin` will just be ignored and no problems occur. But if the question arises the proper answer is available and the program continues. There is an easy way to do this.

In the script which actually lauches the job there will be a line like

```
FPLO 2>/dev/null >out
```

We have to change it into

```
FPLO < ./+yes 2>/dev/null >out
```

or

```
cat ./+yes | FPLO 2>/dev/null >out
```

where the file `+yes` has to be created before (by the script) in the directory, where FPLO is running. This file contains a single line with the proper answer ('y' to skip the split in allmost all cases or 'n' in the particular situation, which we discussed above.). The file may be created with a `here-script` like in

```
cat <<EOF > ./+yes
n
EOF
```

or with `echo` like in

```
echo "y" > ./+yes
```

So, comming back to our situation. To enforce the repeated spin split we would (in the script) switch on the `inital polarization` and we would put an 'n' into the file `+yes`. After a succesful split (means after the re-started calculations passed the splitting), we should imediatly replace the 'n' by an 'y' and and switch off the `inital polarization` in the script, to avoid an unwanted destruction of the now hopefully correctly converged results.

In fact one should allways use this construct, just in case, that one forgot to switch off the `inital polarization`.

Some people will know the unix command `yes`, which will do a similar job as the file construct. However, in practical applications we often met the situation, where this command was not available in a job-queue environment. Therefore, we decided to use the file trick.

5.5 Example

We give an example script here, which uses basic features explained above. It is contained in the distribution directory (in case of default installation something like `FPLO/FPLO14.00-45`) under `DOC/Scripting_example/`.

```
#! /bin/sh
#
# Example script to create a series of calculations for varying lattice
# constant for fcc Al.
#
# We use the so called here-script mechanism to create the pipe input
# on the fly.
# This script works with bash at least. For other shells part of the syntax
# may be different. Consult your man pages.
#
# We assume that the script is exectuted in a directory, where there is
# an inital calculation sub directory called SC containing a converged
# calculation of the same compound (fcc Al).
#
#########################################################################


# Always use fully qualified names, to assure proper program version.
# Store the exec names in variables, so we do not need to scan the script
# to replace the version later on.
FEDIT=fedit14.00-45-x86_64
FPLO=fplo14.00-45-x86_64
```

```
# Give all directories a name prefixed with the name of the parameter,
# which is running, followed by the parameter itself
prefix='a0='

# Remember the current directory.
# Be aware of the back quote syntax, on some unix systems you need a different
# construct to cast the output of a command (pwd here) to a string.
ROOT=`pwd`


##############################################################################
# some functions
usage() {
  echo "usage:  $1 [-r]  [-h[elp]]"
}

##############################################################################

# Check the command line flags.
#
RUN_IT=0

while :
  do case $1 in
    -r) RUN_IT=1
        shift 1
        continue
        ;;
    -h*)
        usage `basename $0`
        exit
        ;;
    -*) usage `basename $0`
        echo 'wrong options'
        exit
        ;;
    *)  break
        ;;
  esac
done
```

```
#
# security questions
#
if [ x$RUN_IT = x1 ]
then
    echo "Shall I run the jobs: [y/n]" ; read YN
    # The next test is a little trick to circumvent problems with
    # empty variable. We first form the concatenation x$YN, which
    # has the value of $YN prefixed with a single x.
    # Then we compare it with the teststring (y in our case) prefixed by x.
    # So, if $YN=y than also x$YN=xy. Disturbing?
    if [ "x$YN" != "xy" ] ; then
        echo "abort"
        exit
    else
        echo "Will run jobs now."
    fi
else
    echo "Shall I (re)create the input: [y/n]" ; read YN
    if [ "x$YN" != "xy" ] ; then
        echo "abort"
        exit
    else
        echo "Will (re)create input  now."
    fi
fi

##############################################################################




# loop over the running parameter, in our case the lattice constant
for xx in 6.50 7.00 7.50 8.00 8.50
do

    # make sure we are in the root directory of our data directory tree
    cd $ROOT

    # create the directory name as described above (example 'a0=6.00')
```

```
    dir="$prefix$xx"


    # check, if input creation or job running shall take place
    if [ x$RUN_IT = x0 ] ; then
        # input creation branch


        # if the directory does not yet exist, create one
        if [ ! -d $dir ]
        then
            mkdir $dir
            echo "directory $dir created"
            # copy the essential files from the inital calculation
            # which is assumed to be in the directory SC
            cp  ./SC/=.sym ./SC/=.dens $dir
            echo " =. files copied"
        else
            echo "directory $dir exists allready"
        fi


        # change into the directory of paramter $xx
        cd $dir

        # Now create the pipe file content, with the help of a here-script.
        # For the sake of book keeping we will save the pipe info into a file.
        # (One could equally pipe directly into fedit.)



        # The next command is the here-script ( <<EOF ), whose stdout is
        # redirected ( > ) into the file ./=.pipe.
        # Every thing which comes between the <<EOF line and the line below,
        # starting with EOF, will go into ./=.pipe. The main advantage of this
        # approach is, that we may use shell variable replacement
        # (interpolation) to put the information of the running variable
        # $xx at the proper position
        cat <<EOF  > ./=.pipe
###############################################################################
# this is the beginning of the pipe file
```

```
# go to symmetry menu
@+@
    # title
    @c@Al, a0-variation
    # enter spacegroup select box
    @s@
        # select space group
        @225@
        # leave selectbox
        @x@
    # structure type
    @t@
       # crystal
       @c@
       # leave select box
       @x@
    # lenth units
    @u@
        # bohr radii
        @b@
        # leave select box
        @x@
    # lattice constants; Here we put our running variable.
    # We enter as first lattice constant the value which is in $xx,
    # then we use the fedit-','-syntax to repeat the value
    @l@ $xx , ,
    # set axis angles
    @a@90.,,
    # setup Wyckoff positions
    # number is one in our case
    @n@1
    # Now, give list of ALL !!! Wyckoff positions.
    @1@ Al @ 0.,,
    #
    # NOW CALL UPDATE, NEVER FORGET THIS!!!
    #
    @+@
    # leave symmetry menu
    @x@
# back in main menu
# This was the symmetry setup, and now we follow our advise to create
# the default =.in input by using the REBUILD-action.
```

```
# (The space before the 'e' opens the alternative menu bar.)
@ e@
# now we have the default input, and are still in the main menu

# Let us set the number of k-points to a non default value:
@k@ 16,,

# Let us set the xc-potential version now:
# first enter select box
@v@
    # select via search
    !Perdew Wang 92!
    # leave select box, go back to main menu
    @x@

#Let us set the relativistic mode:
@r@
    # Select by search, please note that the parentheses indicating the hotkey
    # are not considered in search mode.
    # (Have a look at the select box interactively.)
    !scalar relativistic!
    # leave select box
    @x@



# Let us set some options:
# enter options menu
@-@
    # Good habit is to select all options explicitly
    # This includes the options, which are selected by default.

    # here an example for deselecting
    !PLOT_BASIS!-
    # here an example for selecting,
    !NO_SYMMETRYTEST!+

    #leave menu
    @x@


# Let us select spin=1 explicitly:
```

```
@s@1

# Let us switch off inital polarization explicitly:
@i@f









# last action must be
@q@

# this is the end of the pipe file
#########################################################################
EOF


        # Now execute fedit in pipe mode and use the information of the file
        # =.pipe just created in $dir.
        # We made sure that we are in $dir, since fedit shall act there!
        # We explicitely tell fedit to use the fplo executable stored in the
        # variable $FPLO to have a well-defined state.

        $FEDIT -p $FPLO -pipe <./=.pipe 2>./+log 1>/dev/null

        # Check exit/return code of fedit, there must not be any command
        # in between the check and the command, which produced it (fedit here).
        # The return code is stored in the variable $? in shell.
        # exit=1 means success
        if [ $? -ne 1 ]
        then
            cat<<EOF
Content of log file:
-------------------------------------------------------------------------
EOF

            cat ./+log

            cat <<EOF
-------------------------------------------------------------------------
There was an error in the pipe input. Check logfile above or in $dir/+log.
```

```
Be aware that the line numbers refere to the file $dir/=.pipe!
EOF
            exit 2;
        fi

        # If we are here, the input was created in $dir according to our setup
        # Now we continue with the next parameter

        # change back to where we started
        cd $ROOT


        # end of input branch

    else

        # job-run branch

        # change into the directory of paramter $xx
        cd $dir

        echo "$FPLO running in  $dir ..."

        # now execute, whatever is nessecary to launch job in the current
        # directory (name $dir)

        #START: example
        # We just run the jobs sequentially on a single machine
        # and redirect stdout to file out and stderr to /dev/null.
        # (In this way there will be no dangling output and the job could run
        # savely in background, which is not done in our example here.)

        # Furthermore, we use the +yes-file mechanism to avoid a crash
        # due to repeated inital polarization (spin split).
        # The "y" below enforces fplo to continue in such situation
        # without a repeated split and  does nothing otherwise. See manual.

        echo "y" > ./+yes


        cat +yes | $FPLO 2>/dev/null > out
```

```
        #END:   example



        # change back to where we started
        cd $ROOT
    fi


# end of xx-loop
done

if [ x$RUN_IT = x1 ] ; then
    grepfplo -p 'a0=' -m EE | tee e
fi

#
# After the input creation run we should have a directory structure like
#
# ./SC/
# ./a0=6.00/
# ./a0=6.50/
# ./a0=7.00/
# ./a0=7.50/
# ./a0=8.00/
# ./a0=8.50/
# ./script
#
# where every directory contains the same setup, except for the
# lattice constant.
# We may now perform converged calculations (option -r) in all directories.
# If we want to change, say, the number of k-points, we edit this number
# in the pipe-section above, re-run that script to change the input and
# re-converge the calculations (option -r).
#
```

To test the script, copy the directory Scripting_example (including the subdirectory SC) to an appropriate place, change into the copy and edit the variables FPLO and FEDIT in fscript to point to the fully qualified executable names of your instalation. Make sure that the PATH is properly set, so that the executables may be called in your environmet.

Then execute

```
./fscript
```

and answer 'y'. The directory structure described at the end of the script should have been

created now. Next execute

```
./fscript -r
```

and answer 'y'. Now the calculation is running in sequence. Wait for it to finish. Test convergence via

```
grepfplo -p a0= -m it
```

Check termination status via

```
grepfplo -p a0= -m term
```

Collect total energies in file named 'e' via (done in the script with option `-r`)

```
grepfplo -p a0= -m EE | tee e
```

Plot file 'e' with e.g. XFBP if you want to.

The old scripts 'gr..' are part of the installation and should be available if your installation procedure succeded. But now you can use GREPFPLO instead.

Appendix

A Summary of Changes

FPLO14:

1. **Internally a correction for diverging non-spherical density parts in scalar relativistic calculations was added, which leads to slightly changed total energies as compared to previous** FPLO **versions.**
2. Multiple compilation branches possible (`install/MMakefile 1`). See Sec. 12.
3. New tools, see Sec. 9.
4. The basis can be changed. 18
5. Density mapping: Sec. 4.4.28
6. New input files, some changes of old input files into fedit menues: Sec. 3
7. Band unfolding: Sec. 4.4.16
8. Molecular/Individual band weights: Sec. 4.4.25
9. Optics: Sec. 4.7
10. Output of energy and band resovled real space densities/wave functions for plotting. (See FEDIT `GRID OUTPUT` submenu)
11. Z_2 invariant for topological insulators with inversion centers. This can be found in an FEDIT submenu.
12. Reduced exchange field (LxSDA, GGAx). This can be found in the FEDIT menu.
13. Fixed-spin-moment works now in full-relativistic mode. Note, that this is not as stable as in non-full-relativistic settings due to the fact that spin is not a good quantum number in full-relativistic mode, which requires an iterative scheme for full-relativistic FSM.
14. The charging has enhanced options. Now, one can use Virtual Crystal Approximation (VCA), molecular ionicities (for molecules) and a jellium model for 3D-solids. The old files `=.atcharge` and `=.mol_charge` have been removed and placed into an FEDIT submenu. Data of these files are transfered to `=.in` on version update.
15. The production of `bravais.ps` and `primitive.ps` is discontinued, due to the availability of XFPLO.

16. The file `=.coeff` has been turned into the switch `Output +coeff file` in the FEDIT bandstructure submenu.
17. Full relativistc LSDA+U (but still very bad convergence)
18. Note, that we changed the table for the onsite orbital momentum to show entries for each site not each sort. This is repetitive information, but more consistent with other tables.

2009:

1. Geometry optimization of internal parameters (Wyckoff positions) via forces.
2. Flexible grid output of density, spin-density and potential for visualization. (Now, in an FEDIT sub-menu and with `opendx` output.
3. Increased performance for larger cells and heavier atoms.

2008:

1. Generalized Gradient Approximation (GGA).
2. Scalar relativistc version in Koelling Harmon style
3. Finite nucleus has been implemented for upcoming use with EFG calculations.
4. Re-implementation of VCA.
5. Test version of Wannier functions implemented.

2007:

1. Currently the basis is predefined. The user is not expected to have to change the basis.
2. The basis is fixed, therefore the old basis optimization is discontinued.
3. New potential construction and local orbital integration with higher accuracy.
4. Molecule/Cluster extension.
5. Major rewrite of the code.
6. Forces, but not geometry optimization modeul yet

01.07.2005:

1. file `=.ldos` is obsolete now, added to FEDIT menu
2. automatic Ewald parameter and Fourier component choice implemented
3. Orbital moment output for full-relativistic calculations
4. **Experimental:** Core-confinement for $4f$-systems
5. minor changes in atom potential to increase stability.

01.07.2004:

1. Full relativistic mode implemented (not for CPA and LSDA+U).
2. New internal mesh settings for increased accuracy.
3. Accuracy improvements in various places.
4. CPA and LSDA+U work together (not full relativistic).
5. Basis orbital grouping implemeted for enhanced stability and avoiding of fixed x_0.
6. Large-scale code restructuring in symmetry treatment in order to implement the full relativistic mode.
7. Format of `=.dens` hanged.

03.03.2003:

1. New shape function implemented, which should make voronoi cell merging obsolet. This has an influence on the total energy within the mHartree range.
2. New spherically averaged crystal potential (orbital calculation) introduced, which now is the default. This changes the basis definition and therefore the values of the optimum x_0. It results in a change of total energies. For 3d metals the new basis is a bit worse than the old. However, inclusion of 4d polarization states will, as before, converge the basis to a high degree. For more open structures like oxides the new basis scheme is more stable and in many cases better.
3. CPA implemented.
4. LSDA+U implemented.

31. Oct. 2002:

1. A single step calculation (niter=1) will not overwrite the `=.dens` and `=.basis` files
2. A bug in the symmetrytest is fixed.
3. Enhanced accuracy in the twocenter integrals.

10. Jun 2002:

1. Change in scalar relativistic definition. (Tiny total energy differences < 1 mHa expected.)
2. Kinetic energy correction due to finit radius of orbitals added. Changed results expected, if the basis contains orbitals with large non-vanishing derivative at the outermost mesh point.

06. May 2002:

1. Bugfix for COMPAQ-fort Linux-alpha compiler in bzone.f90.

12. Mar 2002:

1. Definition of empty sites changed: The addition of empty sites to a normal structure will no more change the basis definition of the normal atoms. Thus, the addition of empty sites has to lower the total energy (which was not the case before)!
2. Total crystal density may be output using a grid definition file `=.densgrid`. See Section 3.

B FAQ

Q: How can one use existing input files with different versions of FPLO?

A: Updating is simple, just call the newer FEDIT. It will update the input files. Then continue with the corresponding FPLO.
For downgrading see Chapter 2.

Q: I started FEDIT and got the message

```
error(ReadPCTable): Cannot find entry definition file
 '+fedit'!
 It should be in the local tmp directory '+tmp'!
 One can overwrite the location using option -ef filename.
 Possible reason:
        1) the executable '...fplo...' is not running correctly
           or just does not exist!
        ....
```

A: Make sure that the FPLO executable (of the same version as FEDIT) is within the `PATH` (shell environment variable). Another possible source of the error is that the directory or some of the relevant files are write protected. If the FPLO executable is really not running (some linked libraries not found), just run FPLO by hand on command line and see what happens. Be aware of the fact that FEDIT will automatically try to execute the FPLO executable using the fully qualified name (with the 'version-release' and architecture-suffix, like `fplo14.00-45-x86_64`). It will not use the generic name `fplo`.

Q: How can I save memory?

A:

1. The number of `occupied bands` can be specified in the FEDIT main menu. Read the help screens for this variable. This does **not** work for CPA.

2. For large compounds the number of k-points in the Brillouin zone may be reduced. (Convergence with the number of k-points has always to be testet!)

Q: I want to make an antiferromagnetic calculation...

A: You will need two `spin sorts` and have to set the `initial polarization` to 't'. Then there is a submenu (press <SPACE> `i`), which allows to set the value of the `inital spinsplit`. Choose the same absolute value with opposite sign for the atoms, which are antiferromagnetically ordered. Set zero for atoms, which by symmetry must have zero moment. To assure zero total moment you may use the FSM method with a total `spin moment` of zero.
Note, that in the current implementation equivalent atoms with opposite magnetic moments have to be put on different Wyckoff positions. This is one of the rare occasions where the code does not make full use of symmetry.

Q: I want to plot the real space density or potential...

A: Use the grid-oputput sub-menu.

Obsolete since version 9.00. See description of file `=.densgrid` in Chapter 3. The output is performed after the potential calculation in each iteration cycle (otherwise the potential is not yet defined). Use a plot program of your choice to visualize the data (e.g. opendx).

Q: FPLO cannot allocate memory.

A: Ask the system administrator to increase the limits of stack, heap and data size of a user job. FPLO jobs like other band structure programs need a lot of memory. The memory consumption is proportional to the number of k-points in the irreducible Brillouin zone, which in turn usually is proportional to the inverse of the number of atoms. You should check the number of k-points you really need to solve the physical question.
Use number of `occupied bands` to decrease memory usage!

Q: I rebuilt the code and gave the executables non default names. How can I run FEDIT with the new code.

A: Use an option:

```
FEDIT -p name_of_your_executable
```

This will force FEDIT to work with the executable named after -p. The executable can be an FPLO, BANDPLOT or DIRAC executable. Off course, the executable must be accessible within the current `PATH`.

Q: How can I see the progress of the calculation?

A: Assumed that you re-directed the FPLO output into a file, use the unix command:

`grep "last deviation" fplo_outfile` (for the self consistency)
`grep "dev=" fplo_outfile` (for the force iteration)
XFPLO `-oi` (set output file name in interface if needed)

Q: I want to use `less`, `grep`, `vi` or other unix tools to work with the FPLO-files having a name '`+...`'. It does not work!

A: Many (new) unix tools will interprete the '`+`' sign as an option flag. To use these tools with '`+`'-files, specify `./+file` instead `+file` on command line!

Q: I have started a spinpolarized calculation but the result is not spin polarized.

A: The symmetry between both spin directions has to be broken by hand. Set the `initial polarization` to '`t`'. You can set the amount of the initial split in the alternative submenu `Initial spinsplit`.
Attention: If there are already two spin sorts in the density file, the program will ask if the splitting shall be skipped (which is correct in most cases). However, in the case discussed in paragraph 5.4.1 a resplit was intended. If the job runs in background, it will abort as soon as the question has to be answered since a background job has normally no standard input to read from.

Q: I have started a spinpolarized calculation with initial spin split and the iteration jumps around.

A: May be the attractor regions for the polarized and for the non polarized solutions are selected in consecutive iteration steps and so it jumps. Depending on the given situation

1. one may try to increase the initial spin split.
2. the iteration-mixing may be too large. Got to alt-submenu `iteration` and decrease it.
3. use FSM to preconverge near the expected moment and release the FSM condition after convergence of the FSM calculation.
4. use FSM to scan to whole $E\,(M)$ curve (expensive, but sometimes the only way).

Index

band weights, 10, 15, 17, 18, 23, 33, 51, 115

files
- +atpot..., 25
- +band..., 15, 17, 23
- +basis, 22
- +bweights..., 15, 17, 18, 23, 33–35
- +coeff, 23
- +dens..., 25
- +dos..., 22
- +error, 23
- +fcor..., 24
- +fedit..., 11, 25
- +fkval..., 25
- +fval..., 25
- +grid..., 25
- +har..., 25
- +idos..., 22
- +imeps, 22, 89–92
- +(i)ldos.site.nl, 22
- +loi, 22
- +plasmon, 26, 89–91
- +points, 24
- +run, 23
- +symmetry, 24
- +tmp, 28
- =.addwei, 18, 35
- =.atcharge, 19, 115
- =.basdef, 18, 103
- =.basis, 23, 101, 102, 117
- =.bwdef, 18, 24, 34, 35, 51
- =.cmd, 19
- =.coeff, 19, 116
- =.dens, 15, 17, 23, 53, 54, 102, 117
- =.densconvert, 13
- =.densgrid, 20, 27, 118, 119
- =.densmap, 18, 53, 54
- =.dmat_init, 17, 87, 89
- =.in, 13, 16, 53, 101, 102, 115
- =.kp, 17, 24
- =.ldos, 116
- =.mol_charge, 115
- =.pipe, 98
- =.sym, 11, 16, 53, 102
- =.unfold, 18
- =.xef, 18, 52
- =.xfp, 18
- =.xstr, 18, 52
- bravais.ps, 115
- dmatedit.ini, 27, 89
- grid_dens...[net/general/cfg], 27
- primitive.ps, 115
- tmp, 28

getting help, 1, 10, 37, 54

initial spin polarization, 103, 119, 120

optics, 10, 22, 26, 89, 93

programs
- dirac, 10, 25, 120
- dmatedit, 10, 17, 27, 29, 87–89
- faddwei, 10, 18, 33, 35
- fdowngrad.pl, 13
- fedit, 10
- foptics, 10, 89, 91, 92
- fout2in, 16, 18
- fpiotest, 1, 10, 11

fplo, 9
grepfplo, 10, 30, 114
xfbp, 9, 10, 15, 18, 19, 23, 24, 54, 73, 86, 89, 91–93, 114
xfplo, 9–11, 16–18, 23, 24, 37, 51–54, 115, 120

unfolding, 18, 23, 45